Cloud PIE

Proactive and Intelligent Ecosystem

Trabajo de Fin de Máster

Curso 2017-2018

Autor

Rodrigo Crespo Cepeda

Director

José Luis Vázquez Poletti

Master en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Cloud PIE

Proactive and Intelligent Ecosystem

Trabajo de Fin de Máster

Departamento de Arquitectura de Computadores y Automática

Autor
Rodrigo Crespo Cepeda

Director
José Luis Vázquez Poletti

Convocatoria: *Septiembre 2018*
Calificación: *9*

Master en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

26 de septiembre de 2018

Autorización

El abajo firmante, matriculado/a en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Cloud PIE", realizado durante el curso académico 2017-2018 bajo la dirección de José Luis Vázquez Poletti en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en internet y garantizar su preservación y acceso a largo plazo.

Rodrigo Crespo Cepeda

26 de septiembre de 2018

¡A quemar la pista!

Lúcio Correia dos Santos

Agradecimientos

A José Luis, por guiarme en este recorrido y no dudar de mi trabajo ni un momento.

A Meriem, por su infinita ayuda, ya que sin ella no habría podido lograrlo.

A mi hermana Rebeca, por ayudarme a entender mis palabras, y al pequeño Héctor, por permitírselo.

A mi familia, por estar siempre dispuestos a escucharme y motivarme.

A ti, querido lector, por no aburrirte y dejarlo en esta línea.

Resumen

No ha pasado tanto tiempo desde la creación de los primeros ordenadores y las primeras redes de comunicación de computadoras. Sin embargo, la rápida evolución de la tecnología y la gran asimilación, en los últimos años, por parte del público en general ha motivado la creación de servicios de aprovisionamiento de infraestructura en la nube por suscripción.

La gran masificación de los servicios de cloud computing ha ocasionado que los proveedores de infraestructura experimenten una escasez de recursos que afecta directamente a los consumidores, haciendo necesaria la creación de SLAs para asegurarlos. Esto ha provocado que se busquen nuevas técnicas para realizar un mejor aprovechamiento de los recursos.

Cloud PIE es un ecosistema proactivo e inteligente en la nube, cuyo objetivo es asegurar la disponibilidad de recursos para servicios cloud de la manera más eficiente posible. Para ello, dispone de una serie de módulos, en forma de microservicios, que le permiten tomar decisiones inteligentes sobre las posibles mutaciones que se deben realizar sobre el sistema.

Cloud PIE utiliza un sistema basado en reglas para la toma de decisiones, herramientas de control de infraestructura y obtención de datos de monitorización hardware y software, así como un módulo que le permite evolucionar su conocimiento para tomar decisiones más acertadas y ajustadas a los servicios con los que trabaja.

De esta forma, Cloud PIE se convierte en una tecnología efectiva de gestión automatizada, inteligente y proactiva de los recursos de infraestructura cloud para servicios heterogéneos, como una idea novedosa para dar solución a este problema.

Palabras clave

computación en la nube, sistemas distribuidos, arquitectura de computadores, devops, microservicios, aprendizaje máquina, inteligencia artificial, motor de reglas, infraestructura como servicio, automatización

Abstract

It has not been that long since the creation of the first computers and communication networks. However, in the recent years, the rapid evolution of technology and its great acceptance by the general public, has motivated the creation of infrastructure as a service in the cloud providers.

The growing mass of cloud computing services has caused infrastructure providers to experience a shortage of resources that affects consumers directly, making it necessary to create SLAs for assurance. This has influenced the increase of research in new techniques for resource use optimization.

Cloud PIE is a proactive and intelligent ecosystem in the cloud, whose main goal is to ensure the availability of resources for cloud services in the most efficient way possible. For this purpose, it is composed by a series of modules, in the form of microservices, that allow it to make intelligent decisions about the possible mutations that must be applied to the system.

Cloud PIE uses a rules based system for decision making, infrastructure management and hardware and software monitoring tools, as well as a module that allows it to evolve its knowledge, to make more accurate and adjusted decisions based on the deployed services.

In this way, Cloud PIE becomes an effective technology for automated, intelligent and proactive management of cloud infrastructure resources for heterogeneous services, as a innovative idea to solve the mentioned problem.

Keywords

cloud computing, distributed systems, computer architecture, devops, microservices, machine learning, artificial intelligence, rules engine, infrastructure as a service, automation

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Preámbulo

1.1 Introducción

Cada día estamos más conectados a internet y al gran abanico de servicios y aplicaciones existentes en la nube; y ya no somos capaces de imaginar un mundo en el que la red no forme parte de nuestras vidas. El crecimiento del uso de dispositivos y gadgets móviles, las redes sociales, los videojuegos online, así como la existencia de un gran interés en conectar todos los aparatos electrónicos con la red, el llamado internet de las cosas o IoT, ha provocado que en los últimos años se haya dado una gran evolución en las estructuras de computación en la nube.

La computación en la nube o cloud computing es un modelo que permite habilitar acceso bajo demanda a una serie de recursos de computación configurables y compartidos en la red, que pueden ser rápidamente suministrados con un esfuerzo mínimo y sin necesidad de interacción por parte del proveedor [27].

El cloud computing se ha convertido en un paradigma esencial en nuestro día a día y, es que, aunque no nos demos cuenta, el elevado consumo de servicios en la nube está siendo mantenido por este paradigma, soportando grandes cantidades de información, como

se puede observar en la figura 1.1.

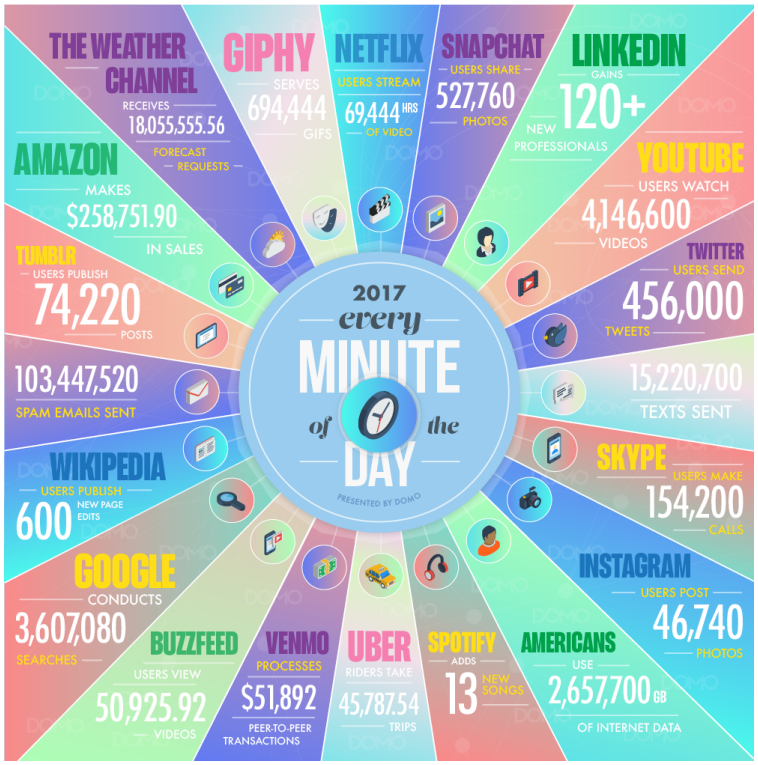

Figura 1.1: *Uso de datos de cloud computing en la actualidad*

Según Vazquez-Poletti et al. [41], el crecimiento en la demanda de la nube da lugar a en una escasez de recursos que afecta tanto a los proveedores como a los consumidores. Esto ha hecho que los proveedores de servicios cloud necesiten utilizar algoritmos de control de admisión a los recursos para optimizarlos, reducir costes y poder soportar la demanda.

Actualmente, las infraestructuras cloud públicas como Amazon AWS[1], Microsoft Azure[2], IBM Softlayer[3], y las de ámbito privado como OpenNebula[4] y OpenStack[5], disponen de sistemas de gestión que permiten solicitar más recursos en caso de que sea necesario, con herramientas sencillas y una gran variedad de opciones de selección.

[1] https://aws.amazon.com
[2] https://azure.microsoft.com
[3] http://www.softlayer.com
[4] https://opennebula.org
[5] https://www.openstack.org

Sin embargo, los recursos no son ilimitados y, por este motivo, los proveedores de servicios públicos disponen de acuerdos de nivel de servicio, Service Level Agreement o SLA, en los que establecen los recursos mínimos que aseguran al contratar sus servicios. Un ejemplo de SLA es el ofrecido por Amazon AWS que se compromete a proporcionar una disponibilidad del 99,95 % durante todo el año. Si sus servicios permanecen sin estar disponibles durante más de 4,4 horas, realizan una devolución del 10 % de la factura del mes al cliente [42].

Uno de los problemas que, según Vázquez-Poletti et al. [40], se han visualizado en los últimos años, y que puede ser fatal para el futuro de la computación cloud, es que los proveedores de servicios en la nube están demasiado orientados a infraestructura y no disponen de capacidades avanzadas orientadas a servicios, que aseguren su elasticidad, la calidad del servicio y dispongan de controles de admisión. Por ello, en los últimos años, se han elaborado estudios que buscan nuevas técnicas para mejorar la eficiencia de los recursos, que se orientan a estudiar las posibilidades creando estructuras intermedias entre los servicios y los proveedores de infraestructura para reducir la carga de trabajo que aplican sobre sus sistemas [20].

En esta situación, la administración de sistemas, ha buscado adaptarse a las faltas que tiene la infraestructura en la nube a la hora de generar servicios y controlar la distribución de estos entre múltiples máquinas, centrándose mucho en simplificar la vida a los desarrolladores. Por ello, se ha creado el nuevo concepto de DevOps [4], que busca unificar el desarrollo de software con la gestión de los sistemas en cloud, mediante sistemas de automatización y monitorización que permitan reducir los ciclos de trabajo y asegurar la disponibilidad de sus servicios. El funcionamiento de DevOps según Atlassian[6] se puede ver en la figura 1.2.

Además, la evolución en las capacidades de la inteligencia artificial [35], y en específico el aprendizaje máquina o machine learning, ponen a nuestra disposición nuevas herramientas con las que trabajar para mejorar la adaptabilidad de los servicios a la escasez de infra-

[6] https://www.atlassian.com/devops

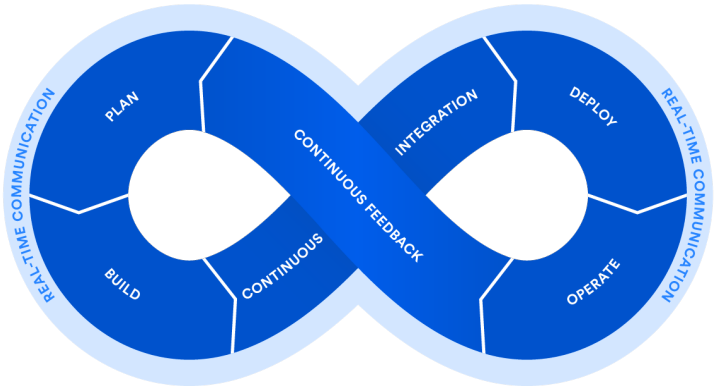

Figura 1.2: *DevOps según Atlassian*

estructura y a la necesidad de aprovisionamiento. Con la inteligencia artificial, podemos mejorar la capacidad de nuestros sistemas de autogestionarse y controlar de una manera eficiente los costes, a la vez que aumentar la disponibilidad de los servicios que habilitamos en la nube. Mediante el reconocimiento de patrones en machine learning [31] en la evolución temporal de los servicios en la nube, se pueden tomar decisiones sobre en qué momentos disponer de más recursos para un servicio específico, cuándo apagar los sistemas o incluso prever posibles cambios, manteniendo una política de prevención de catástrofes.

1.2 Estructura del documento

Para la organización del documento se ha decidido estructurarlo en 6 capítulos que diferencian las distintas etapas que ha sufrido el proyecto desde su inicio.

El capítulo 1, o capítulo de preámbulo, identifica el trabajo de información que se ha realizado previamente al comienzo del proyecto, con la finalidad de identificar la idea y los objetivos del trabajo. Está formado por un apartado de introducción (1.1), esta sección, que informa de la estructura del documento (1.2) y un apartado con los objetivos y el plan de trabajo (1.3) que se han seguido a la hora de realizar el proyecto.

El capítulo 2, o capítulo de investigación, documenta el proceso que se ha realizado para analizar el estado del arte (2.1) y las tecnologías que han sido consideradas y finalmente descartadas (2.2).

El capítulo en el que se describe más a fondo el proyecto es el capítulo 3. En este capítulo se diferencia un primer apartado de descripción de Cloud PIE (3.1), una sección en la que se detalla la arquitectura (3.2) que se ha creado para el proyecto y, finalmente, un último apartado con las tecnologías que se han utilizado (3.3).

Tras la información sobre la arquitectura, se presenta el capítulo 4, que contiene información sobre las pruebas que se han realizado sobre la implementación del proyecto. El capítulo se subdivide en un primer apartado con casos de uso (4.1), un apartado donde se detallan los experimentos (4.2), una sección con un resumen de resultados de los experimentos (4.3), y una última parte en la que se realiza una discusión (4.4) de los experimentos y sus resultados.

El capítulo 5, o capítulo de trabajo futuro, presenta una serie de ideas que se han pensado para dar continuidad al proyecto, posteriormente a la entrega de este Trabajo de Fin de Máster.

Y el último capítulo, el capítulo 6 detalla las conclusiones que se han alcanzado tras realizar el proyecto.

1.3 Objetivos y plan de trabajo

El objetivo principal del proyecto Cloud PIE es la creación de un ecosistema donde un administrador de sistemas sea capaz de mantener sus servicios con alta disponibilidad, adaptándose a cualquier cambio que sea necesario para mejorar la eficiencia del entorno. Por un lado, el sistema debe escalar ante cualquier inconveniente que requiera aumentar los

recursos necesarios para el correcto funcionamiento del entorno.

Se busca evitar que haya momentos en los que el sistema no sea capaz de proporcionar acceso al uso de los servicios por falta de recursos, y reducir al mínimo los retrasos que puedan existir a la hora de aplicar medidas paliativas, ejecutando, si es posible, medidas preventivas que impidan que esas situaciones ocurran.

También, el sistema debería ser lo suficientemente eficiente a nivel de costes, y ser capaz de eliminar las instancias que no estén siendo utilizadas cuando no sean necesarias; además de elegir la opción de recursos que se ajuste a las necesidades y reduzca el gasto general del sistema.

Se pretende que este sistema sea automático y controlable desde un principio. La configuración debe ser sencilla y lo suficientemente completa para que pueda gestionar un gran abanico de aplicaciones heterogéneas. El sistema debe ser capaz de tomar decisiones por sí mismo y de presentar rasgos de inteligencia artificial, aplicando medidas con precisión similar o mejor que las de un ser humano.

Es esencial que Cloud PIE sea multiplataforma y permita trabajar con múltiples proveedores de infraestructura de forma transparente. Uno de los grandes problemas que existen al utilizar sistemas de auto escalado, en estos servicios, es que no facilitan la posibilidad a sus clientes de combinarse con otras plataformas cloud.

El plan de trabajo se basa en realizar fases cortas e iterativas de planificación, desarrollo y testing, utilizando una metodología de trabajo Agile [31], con el objetivo de agilizar el proceso de desarrollo. En cada fase se tratará con diferentes componentes del sistema hasta finalizar el modelo en conjunto. Por último, se realizará una iteración sobre todo el sistema construido y se verificará si cumple los objetivos que se plantean en este apartado.

Chapter 1

Preamble

1.1 Introduction

Every day we are more connected to the internet and to the wide range of existing services and applications in the cloud; and we are no longer able to imagine a world in which the network is not part of our lives. The growth of the use of mobile devices and gadgets, social networks, online videogames, as well as the increasing interest in the Internet of Things, or IoT, has caused a great evolution in the cloud computing structures in the recent years.

Cloud computing, is a model that enables on-demand access to a series of configurable and shared computing resources on the network, which can be quickly supplied with minimal effort and with no need for provider interaction [27].

Cloud computing has become an essential paradigm in our day-to-day life, and even if we do not realize it, the high consumption of services in the cloud is being maintained by this paradigm, supporting large amounts of information, as can be observed in (figure 1).

According to Vazquez-Poletti et al. [41], the growth in the demand for cloud resources results in a scarcity that affects both suppliers and consumers. This has caused cloud

providers the need to use admission control algorithms to optimize resources, reduce costs and be able to support demand.

Currently, public cloud infrastructures such as Amazon AWS, Microsoft Azure, IBM Softlayer, and those of private domain such as OpenNebula and OpenStack, have management systems that allow to request more resources if necessary, with simple tools and a wide variety of selection options.

However, the resources are not unlimited and, for this reason, public service providers have created Service Level Agreements, or SLAs, in which they establish the minimum level of service they can assure. An example of an SLA offered by Amazon AWS, which is committed to provide 99.95% availability throughout the year. If their services remain unavailable for more than 4.4 hours, they make a 10% refund of the customer's month's bill [42].

One of the problems that, according to Vázquez-Poletti et al. [40], has been observed lately and that can be fatal to the future of cloud computing, is that cloud service providers are too infrastructure-oriented and do not have advanced service-oriented capabilities, that ensure their elasticity, the quality of service and admission controls. Therefore, in recent years, there has been a research for new techniques to improve the efficiency of resources, that aim at the creation of intermediate structures between services and infrastructure providers to reduce the workload they apply to their systems [20].

In this situation, the systems administration has sought to adapt to the shortcomings of the cloud infrastructure regarding the deployment of services on multiple machines, focusing to ease the developers effort. For this reason, the new concept of DevOps [4] has been created, which seeks to unify software development with the management of cloud systems, through automation and monitoring systems that reduce work cycles and ensure the availability of their services.

In addition, the evolution of artificial intelligence [35], specifically in machine learning, provide us with new tools to improve the adaptability of services to infrastructure shortages and provisioning needs. With artificial intelligence, we can improve the capacity of our systems to self-manage and efficiently control costs, while increasing the availability of the deployed services. Using machine learning techniques [31], there can be recognized patterns on how the services evolve in time, it can be decided when to have more resources for a specific service, when to turn off systems or even foresee possible changes, by maintaining a policy of catastrophe prevention.

Capítulo 2

Investigación

Tras haber definido los objetivos del proyecto y los problemas que se quieren solucionar con él, se ha procedido a realizar la investigación del estado actual de la tecnología y de proyectos con objetivos similares a Cloud PIE. Ésta es una etapa muy importante para el correcto desarrollo del proyecto, que permite aprovechar y estudiar el conocimiento existente para así poder organizar la arquitectura de manera eficiente y con un nuevo enfoque.

El estudio del estado actual ha ayudado a centrar el ámbito del proyecto, a encontrar tecnologías que pueden ser útiles para el proyecto y descartar otras, y a establecer todo lo necesario para proceder, posteriormente, a implementar el sistema.

A continuación se indican el estado del arte y las tecnologías que se han descartado.

2.1 Estado del arte

En este apartado se describen algunos proyectos que buscan solucionar los mismos problemas que Cloud PIE.

En primer lugar podemos encontrar los sistemas de auto escalado que proporcionan

algunos de los proveedores de infraestructura cloud más conocidos en la actualidad.

2.1.1 Proveedores de infraestructura cloud

Los principales proveedores de infraestructura y contenedores cloud que existen en la actualidad disponen de servicios de auto escalado que aprovechan la información obtenida a través de sus sistemas de monitorización, en combinación con un motor de reglas que permite establecer las condiciones necesarias para que el sistema escale, haciendo de esta forma que sus sistemas sean dinámicos mediante unas reglas preestablecidas.

Un motor de reglas [11], Business Rule Engine o BRE, es un sistema software que se utiliza para definir y ejecutar una gran variedad de decisiones lógicas complejas que son utilizadas por sistemas operacionales. Generalmente son utilizados en investigación y aplicaciones de inteligencia artificial, y suelen utilizar conjuntos de reglas para definir cómo se procesa la información.

Amazon AWS

El proveedor de infraestructura cloud de Amazon dispone de un servicio llamado AWS Auto Scaling que permite al administrador mantener sus sistemas disponibles y realizar escalado horizontal de máquinas [39], es decir, crear máquinas nuevas según las condiciones que defina.

Está especialmente dedicado a ser utilizado con instancias de Amazon EC2[1], pero también se puede utilizar con la base de datos NoSQL de Amazon, Amazon DynamoDB, y con la base de datos relacional compatible con MySQL y PostgreSQL, Amazon Aurora.

AWS Auto Scaling monitoriza el sistema y lo provisiona con recursos adicionales de

[1] https://aws.amazon.com/es/ec2

manera dinámica, con el objetivo de optimizar los costes y la disponibilidad. Se muestra un esquema del funcionamiento de AWS Auto Scaling en la figura figura 2.1.

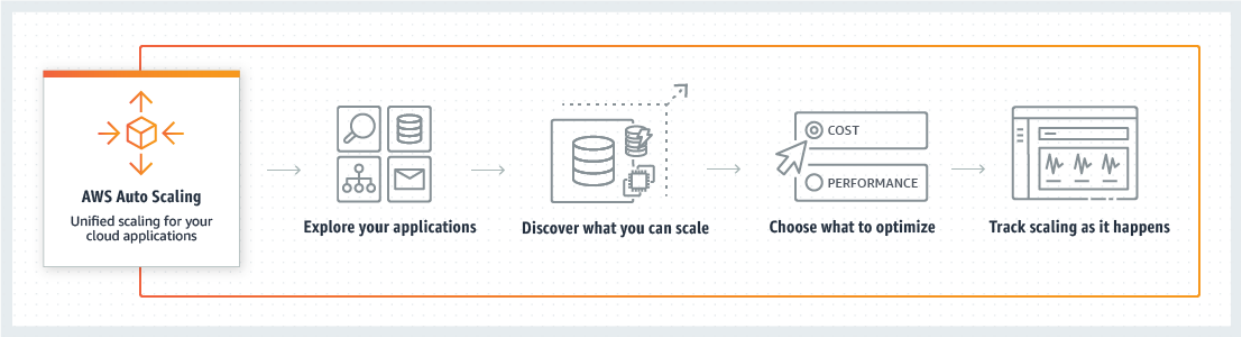

Figura 2.1: *AWS Auto Scaling*

El funcionamiento del motor de reglas de AWS Auto Scaling para controlar las instancias es muy básico. Auto Scaling se basa en la creación de grupos de instancias para el escalado, llamados Grupos de Auto Scaling, que definen conjuntos de máquinas que serán escaladas grupalmente cuando se cumplan unas condiciones específicas, llamadas Políticas de Escalado.

Entre los distintos tipos de Políticas de Escalado, podemos encontrar los tipos tradicionales como el uso de CPU, uso de memoria RAM, etc.

AWS Auto Scaling también permite realizar escalado programado, especificando un momento en el tiempo exacto en el que el sistema debe escalar a un número de máquinas determinado.

Microsoft Azure

Azure, el proveedor de infraestructura de Microsoft, dispone de un sistema de auto escalado llamado Azure Autoscale, con el objetivo de que los administradores cloud tengan un control dinámico del estado de sus máquinas virtuales y aplicaciones.

El sistema de escalado automático de Azure permite realizar tanto escalado horizontal,

como escalado vertical [24], es decir, transformar una máquina virtual en una con mejores o peores capacidades.

Al igual que en el caso de Amazon AWS, Microsoft Azure permite crear grupos de escalado automático e indicar las condiciones que se deben cumplir en los datos obtenidos por la monitorización para transformar la infraestructura. También permite escalado programado.

Algunos ejemplos de uso del escalado automático de Microsoft Azure son:

- Aumentar a 10 instancias los días laborables y reducir a 4 instancias el sábado y el domingo.
- Aumentar una instancia si el uso medio de la CPU es superior al 70 % y reducir una instancia si el uso de la CPU cae por debajo del 50 %.

Google Cloud Platform

Google dispone, al igual que los demás proveedores, de un sistema de auto escalado mediante reglas que permite dar una alta disponibilidad al sistema, provisionando en cada momento con nuevas instancias mediante escalado horizontal. Google Cloud integra directamente este servicio sobre los grupos de instancias que se hayan contratado, y permite establecer políticas basadas en la información monitorizada como el uso de la CPU, la capacidad de servicio de un balanceador de carga [7] u otras métricas obtenidas a través de su servicio de monitorización, Stackdriver Monitoring[2].

Al contrario que los demás proveedores de servicios, Google Cloud no proporciona un sistema de auto escalado programado y espera que el administrador de sistemas gestione esto directamente sobre sus servicios.

[2] https://cloud.google.com/monitoring

Como se puede observar, todos los proveedores de infraestructura proporcionan sistemas de auto escalado configurables y basados en monitorización. Esto confirma la existencia de una necesidad de realizar modificaciones sobre la infraestructura en tiempo real con el objetivo de mejorar costes y disponibilidad. Sin embargo, estos servicios sólo consideran la utilización de sus propios sistemas a la hora de realizar auto escalado, sin ser posible combinarlos con otros proveedores de infraestructura a la hora de gestionar la información y las reglas. Por otro lado, las condiciones que permiten utilizar para controlar el escalado automático, únicamente se encargan de aprovechar los datos de monitorización hardware, sin tener en cuenta la posibilidad de transformar los sistemas basándose en el estado del software que se ha configurado en las máquinas virtuales.

2.1.2 Gestores de contenedores

Los gestores de contenedores son sistemas que permiten automatizar el despliegue, escalado y manejo de aplicaciones de contenedores, como Docker.

Los gestores de contenedores proporcionan funciones para escalar los servicios y contenedores que forman parte del sistema; y en algunos casos, incluyen opciones de auto escalado.

Docker Swarm

Swarm es un gestor de contenedores que está integrado directamente en las herramientas proporcionadas por Docker, al ser éste creación de la misma empresa. Docker Swarm no dispone de un sistema de auto escalado integrado, pero sí que permite utilizar sus herramientas de manera sencilla para escalar los contenedores, aprovechando su propio sistema de monitorización que funciona directamente sobre el hardware de las máquinas virtuales donde ha sido instalado.

Kubernetes

Kubernetes es la solución de Google para la gestión de contenedores. Al igual que con Docker Swarm proporciona herramientas para escalar los contenedores, así como monitorización propia.

Sin embargo, al contrario que Swarm, Kubernetes sí que dispone de un sistema de auto escalado horizontal, llamado Horizontal Pod Autoscaler, basado en unas condiciones definidas previamente.

Kubernetes utiliza los datos obtenidos por su monitorización, como el uso de CPU o métricas personalizadas, para escalar sus contenedores, como se puede observar en la figura 2.2.

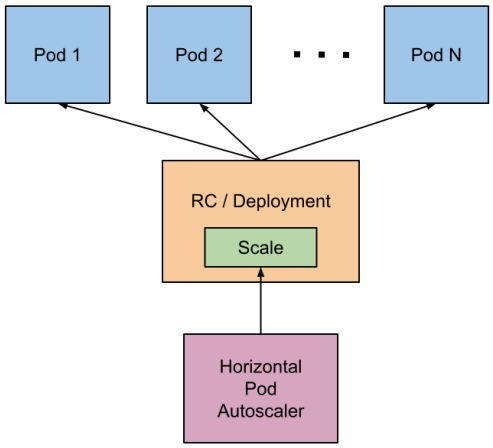

Figura 2.2: *Esquema de la arquitectura de Horizontal Pod Autoscaler*

El sistema de auto escalado de Kubernetes funciona como un controlador en bucle, que tiene una periodicidad especificada por el administrador. En el momento de ejecutarse, el controlador obtiene la información del uso de recursos que ha sido definida en las políticas de escalado, y ejecuta aquellas transformaciones cuyas condiciones hayan sido cumplidas.

Actualmente, en la versión estable de Kubernetes sólo se dispone de la información de uso de la CPU para realizar el auto escalado. Sin embargo, en la versión beta, ya se pueden utilizar todos los valores obtenidos por la monitorización, así como condiciones específicas que permiten aprovechar todo el espectro que cubre Kubernetes para definir reglas de auto escalado, como puede ser la cantidad de veces que se ha accedido a un contenedor específico a través de un balanceador de carga.

Gracias a la existencia de contenedores de aplicaciones, se puede realizar un escalado rápido y un control absoluto de los servicios existentes en un clúster [36]. Estos sistemas permiten que la gestión de escalado sea independiente de los proveedores de servicios con los que se haya contratado la infraestructura. Sin embargo, y pese a que algunos de ellos ya integran sistemas de auto escalado, estos se encuentran en un estado actual de desarrollo y, de nuevo, sólo se centran en la información obtenida del hardware o del software del propio gestor de contenedores. Por otro lado, estos sistemas sólo son capaces de reaccionar en tiempo real, en el momento en el que unas condiciones específicas están ocurriendo, y no son capaces de aprender patrones de funcionamiento.

2.1.3 Proyectos similares

Tras realizar una investigación de los proyectos que se asemejan a Cloud PIE, se han encontrado dos proyectos que tienen muchos puntos en común. Gracias a esto, se ha aprendido de otras soluciones similares para incluir en el proyecto nuevas funcionalidades y mejoras.

Cloud Auto-Scaling

El proyecto Cloud Auto-Scaling[3], creado por Anthony Shaw, pretende detectar cargas de trabajo elevadas en un grupo de sistemas virtuales y usar las API de proveedores de

[3] https://github.com/tonybaloney/Cloud-auto-scaling

infraestructura para lanzar nuevas instancias de manera automática en clústeres con balanceadores de carga. Su intención es crecer con picos de demanda y reducir el número de instancias cuando sea apropiado.

- Soporta múltiples proveedores cloud
- Integración directa con Abiquo para instanciar nuevas máquinas virtuales
- Especificar un clúster con el siguiente perfil:
  - Mínimo y máximo número de máquinas virtuales
  - Plantillas disponibles para las nuevas máquinas
  - Red privada que proporciona direcciones a las máquinas
  - Número ilimitado de triggers
- Un trigger recoge datos y toma decisiones para escalar un cluster basándose en reglas especificadas por el usuario, como pueden ser:
  - Especificar un límite inferior (p.e. Carga media <0.2)
  - Especificar un límite superior (p.e. Carga media >2)
  - Indicar el tiempo que debe estar un resultado sobre el límite antes de escalar
- Aplicación web que permite al usuario manejar los clústeres, los triggers y la API.
  - Ver resultados de datos para un clúster
  - Visualizar un historial de eventos de escalado

Este proyecto, que lamentablemente lleva descontinuado más de 6 años, es muy parecido a Cloud PIE y tiene un espectro objetivo semejante. Sin embargo, se dedica estrictamente a la parte de escalado mediante reglas hardware, sin plantear la posibilidad de que exista un aprendizaje que permita evolucionar las reglas haciendo que el sistema sea más eficiente, ni tampoco la posibilidad de utilizar reglas software.

GridWay

GridWay[4] es un gestor de carga de trabajo que gestiona la ejecución de trabajos y la intermediación con los recursos en un grid formado por plataformas informáticas heterogéneas que se pueden extender con recursos cloud. GridWay permite que la ejecución de los trabajos sea desatendida, confiable y eficiente, al encargarse de programarla en el tiempo y adaptarla a las condiciones cambiantes del grid. Entre las tareas que realiza se pueden encontrar, la capacidad de realizar cambios en la programación de la ejecución, recuperación de errores y migración de los trabajos a otras máquinas. Se puede ver un esquema de la integración de GridWay en un sistema grid en la figura 2.3.

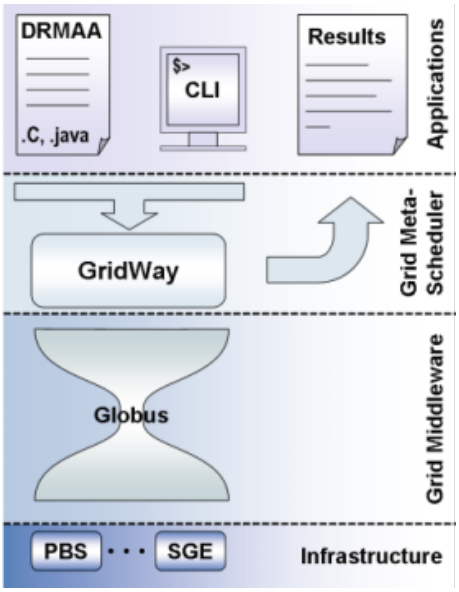

Figura 2.3: *Integración de GridWay en Globus Middleware*

Este proyecto tiene algunas semejanzas con Cloud PIE ya que su objetivo es liberar al administrador de la carga de tener que controlar toda la ejecución de sus procesos distribuidos en máquinas heterogéneas y lo hace con algunas ventajas como la de buscar la mayor eficiencia a la hora de ejecutar los trabajos mediante migraciones y reprogramación

[4] http://gridway.org/doku.php?id=documentation:release_5.4:mad

de tareas.

2.2 Tecnologías descartadas

Durante el proyecto se han probado y se ha planteado utilizar algunas tecnologías que finalmente han sido descartadas. Estas tecnologías son:

2.2.1 Terraform

Terraform[5] [9] es una herramienta software, creada por HashiCorp, que permite construir y mutar infraestructura para entornos de producción de forma segura y eficiente. Es capaz de manejar los proveedores de servicios más populares que existen actualmente, y facilita la integración de soluciones personalizadas para infraestructura propia. Además utiliza el concepto de infraestructura como código, ya que permite generar infraestructura, configurando el sistema a través de un lenguaje de alto nivel.

Terraform utiliza archivos de configuración que describen los componentes que necesita para ejecutar una aplicación en un centro de datos. Con ellos, Terraform genera un plan de ejecución que describe lo que va a hacer para alcanzar el estado deseado y lo ejecuta para construir la infraestructura que ha sido descrita. Cuando la configuración cambia, Terraform es capaz de determinar qué es lo que ha cambiado y crear planes de ejecuciones que modifiquen la infraestructura con los nuevos datos.

Terraform es capaz de manejar componentes de bajo nivel como instancias de computación, almacenamiento y redes; además de componentes de alto nivel como entradas DNS, características SaaS, etc.

Esta librería ha sido utilizada en Cloud PIE durante la fase inicial del proyecto, pero

[5] https://www.terraform.io

finalmente ha sido descartada. Terraform realiza una funcionalidad que es de gran utilidad para la parte de construcción de infraestructura del proyecto, y ayuda a evitar tener que incorporar librerías por cada proveedor para poder incluirlo entre los disponibles. Sin embargo, Terraform requiere un elevado trabajo de integración, al no disponer de librerías que permitan utilizarlo programáticamente de manera sencilla y configurar los componentes sin tener que tratar con archivos de configuración en texto plano.

2.2.2 Vagrant

Tras realizar pruebas con Terraform y descartarlo, se ha intentado probar la otra solución para creación de infraestructura de la empresa HashiCorp, la herramienta Vagrant[6] [33].

Vagrant es el hermano menor de Terraform, ya que su objetivo es la creación de infraestructura en entornos de desarrollo en local y dispone de mucha menos capacidad de configuración. Sin embargo, la gran aceptación por la comunidad de programadores, ha hecho que sea una herramienta muy potente, con gran cantidad de tutoriales y documentación disponibles en la web, así como muchos plugins que permiten que sea utilizado con la mayoría de los proveedores de infraestructura más populares.

Vagrant es una herramienta para construir y gestionar entornos de máquinas virtuales con un flujo de trabajo unitario y fácil de utilizar. Vagrant se centra en la automatización; y eso le permite reducir el tiempo de configuración de entornos de desarrollo, incrementando la paridad con el entorno de producción y haciendo que el concepto de "funciona en mi ordenador" sea una excusa del pasado.

Finalmente, pese a que el sistema de configuración de Vagrant es mucho más simple y hay una gran cantidad de información de cómo utilizarlo, Vagrant ha sido descartado por motivos similares a Terraform. Anteriormente, Vagrant disponía de una librería en lenguaje

[6] https://www.vagrantup.com/

Ruby que permitía utilizar sus funciones directamente de manera programática; sin embargo, esta librería ha quedado obsoleta y ya no es actualizada por HashiCorp, por lo que no se puede utilizar para el proyecto.

2.2.3 Node.js

Node.js[7] [37] es un entorno de ejecución multiplataforma, de lenguaje JavaScript, orientado a eventos asíncronos, diseñado para construir aplicaciones en red escalables. Node.js es de código abierto y es utilizado por una comunidad de desarrolladores muy amplia. Es uno de los lenguajes de programación más populares del momento, como se puede observar en la figura 2.4.

Este sistema es fácilmente instalable los servidores de base Linux más populares, y presenta una gran utilidad a la hora de realizar proyectos en plazos de tiempo muy cortos.

La utilización de Node.js y el lenguaje JavaScript han sido altamente valoradas para el desarrollo de Cloud PIE, ya que la gran comunidad que presenta ayuda a la existencia de librerías muy útiles para el proyecto. Sin embargo, la poca claridad del lenguaje y la necesidad de gestión asíncrona que requiere el trabajo con JavaScript han influido en la decisión de descartar esta tecnología para el proyecto.

2.2.4 Python

Python[8] [38] es un lenguaje de programación de alto nivel, interpretado y orientado a objetos que permite trabajar con mayor celeridad y crear aplicaciones con gran efectividad. Dispone de estructuras de datos de alto nivel y gracias al tipado dinámico y la sustitución dinámica de parámetros es muy efectivo a la hora de desarrollar aplicaciones rápidamen-

[7] https://nodejs.org
[8] https://www.python.org

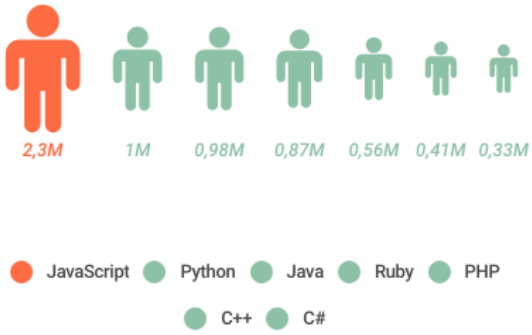

Figura 2.4: *Lenguajes de programación más populares de 2017 en Github*

te. Utiliza lenguaje muy simple y fácil de aprender, con una sintaxis muy centrada en la legibilidad, con el objetivo de reducir el coste de mantener los programas.

Python soporta el uso de módulos y paquetes para estimular la reutilización de código y la modularidad. Una gran ventaja de Python es que está integrado directamente en muchos sistemas operativos Linux populares, lo que facilita utilizar sus programas sin necesidad de mucha configuración.

Todos estos elementos han hecho que Python sea planteado como posible lenguaje base para el proyecto Cloud PIE. Sin embargo, pese a la gran comunidad que se puede encontrar detrás de este lenguaje, muchas de las librerías que utiliza están obsoletas o son incompatibles entre las dos versiones principales de Python, lo que dificulta la gestión del proyecto. Esto ha hecho que este lenguaje de programación sea descartado.

2.2.5 Docker Swarm

Swarm[9][10] [30] es un gestor de contenedores Docker que viene directamente integrado en la herramienta de líneas de comandos proporcionada por el propio Docker. Con Swarm, los administradores de sistemas y los desarrolladores pueden crear clústeres de servicios incluidos en contenedores y gestionar sus nodos mediante un sistema virtual sencillo.

Swarm se distribuye en múltiples máquinas virtuales, los nodos, siendo uno de ellos el principal desde el que se controlan todas las funciones de Swarm. Swarm permite la creación de muchos tipos de componentes que forman parte del clúster, entre ellos: balanceadores de carga, redes, almacenamiento, servicios, contenedores, etc.

Además, Docker Swarm dispone de un set de herramientas que permite visualizar el estado en tiempo real de varios parámetros del hardware de las máquinas virtuales donde ha sido configurado.

Esta capacidad de configuración y la habilidad de proporcionar información sobre el hardware han hecho que el uso de esta tecnología sea planteado en las primeras fases del proyecto para el escalado de los sistemas y la obtención de datos informativos. Sin embargo, finalmente se ha descartado el uso de la tecnología en favor de Kubernetes (sección 3.3.8), otra tecnología similar pero más completa.

[9] https://docs.docker.com/engine/swarm
[10] https://docs.docker.com/get-started/part4/#understanding-swarm-clusters

Capítulo 3

El proyecto

3.1 Descripción

Cloud PIE (figura 3.1) es una herramienta que permite crear un ecosistema en la nube que reaccione de manera proactiva a estímulos detectados mediante sensores de monitorización y produzca, en consecuencia, resultados inteligentes que determinen la evolución del sistema, tomando decisiones automáticas de estructuración y escalamiento de la arquitectura.

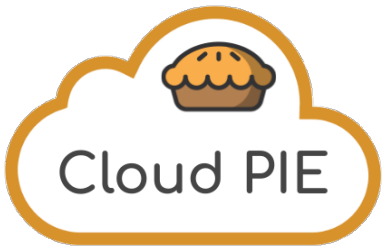

Figura 3.1: *Propuesta de logo de Cloud PIE*

Actualmente, muchas de las tareas relacionadas con DevOps han sido automatizadas mediante herramientas que facilitan al administrador de sistemas la gestión de los servicios en Cloud. Entre ellas, podemos encontrar distintos tipos que permiten crear infraestructura mediante sencillas plantillas, automatizar la configuración y gestionar los servicios que

forman parte del sistema, así como visualizar el estado de la infraestructura.

Sin embargo, a la hora de decidir si el sistema es eficiente nos encontramos con bastantes limitaciones para gestionar las posibles inclemencias que puedan ocurrir, como picos altos y bajos de uso o la necesidad de mayor disponibilidad de un servicio específico. Algunas de las soluciones para este tipo de problemas son el escalado automático mediante reglas basadas en monitorización porcentual del uso de las máquinas físicas, o la provisión a determinadas horas de determinados servicios.

Cloud PIE pretende solucionar estos problemas de una manera más eficiente y automatizada que permita al propio sistema actuar rápidamente basándose en los esquemas tradicionales que se utilizan para gestionar el escalado y la provisión de servicios, aprendiendo además de las distintas situaciones para evolucionar y ser capaz de reaccionar con mayor eficiencia en un futuro. De este modo, si se da la situación de que un suceso es recurrente en el tiempo, el sistema aprenderá que en ese momento, y cuando se dan las condiciones necesarias para que ocurra esta situación, tiene que haber previsto la aparición del suceso y actuar en consecuencia sin necesidad de que intervenga el administrador ni de solucionar el problema a posteriori; tratándose así de un proceso invisible que mejora la eficiencia y evita posibles pérdidas al no tener que esperar a que el problema ocurra.

3.2 Arquitectura

El proyecto se compone de 5 módulos de trabajo o workers, que realizan funcionalidades unitarias, y de un módulo maestro, el cerebro o brain, que unifica y hace uso de los demás para dar utilidad al sistema, como se muestra en la figura 3.2. Todos los componentes están organizados en forma de microservicios para que puedan ser integrados fácilmente en estructuras de contenedores y dividir el trabajo en pequeñas partes.

Según Newman [32], los microservicios son servicios pequeños y autónomos que trabajan juntos. Sus beneficios son muchos, y entre ellos podemos encontrar los siguientes:

- Heterogeneidad
- Resistencia
- Escalabilidad
- Facilidad de despliegue
- Capacidad de organización
- Capacidad de composición
- Optimización frente a reemplazo

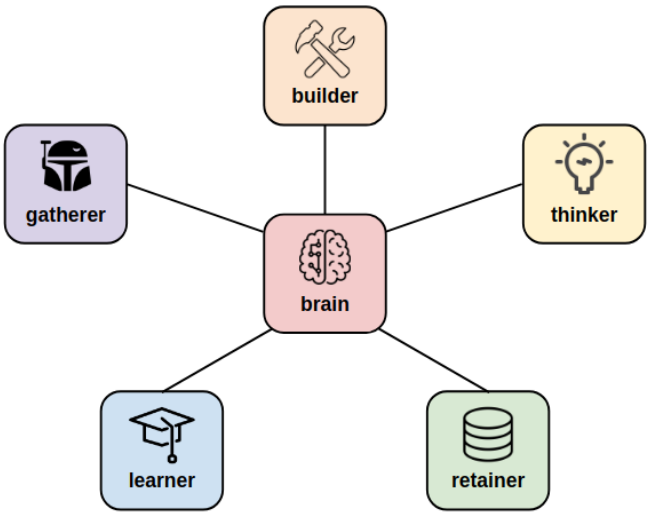

Figura 3.2: *Arquitectura de Cloud PIE*

Gracias a ello, los workers funcionan como módulos simples que realizan funciones unitarias y que son capaces de trabajar por separado. Esto permite que se puedan integrar en

otros sistemas más complejos, actualizar su funcionamiento sin afectar al resto de módulos o incluso eliminarlos si hiciera falta en algún momento para sustituirlos por otros módulos más eficientes. Además, en caso de que una de las funciones tenga una carga muy elevada de trabajo, la distribución en servicios unitarios permite escalar y replicar el servicio para dar más disponibilidad al sistema. Esta modularidad también permite que puedan ser utilizados sólo algunos de los workers para determinadas tareas, como pueden ser la recolección de datos de monitorización, el almacenamiento en bases de datos, la toma de decisiones, el aprendizaje de rutinas o el escalado de máquinas virtuales.

A continuación se detalla la descripción de cada módulo worker: Gatherer, Thinker, Builder, Retainer y Learner.

3.2.1 Gatherer

El módulo de recolección o monitorización de datos es uno de los módulos de trabajo base y es esencial para el sistema, ya que es el que se encarga de obtener toda la información relativa al estado actual del ecosistema mediante una serie de sensores de monitorización hardware y software, como se puede observar en la figura 3.3.

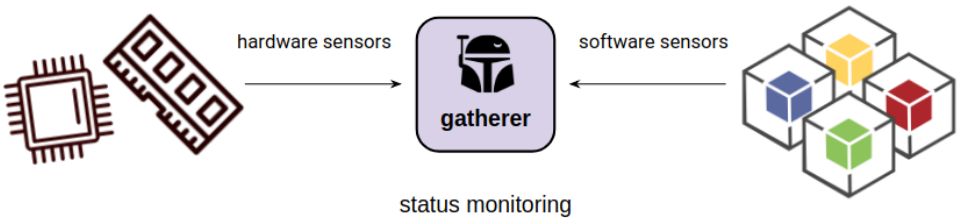

Figura 3.3: *Esquema del módulo Gatherer de Cloud PIE*

Sensores hardware

Esta funcionalidad del módulo proporciona información sobre el hardware de la infraestructura sobre la que se levanta el sistema, entre la que se pueden encontrar datos principales como:

- Uso de CPU
- Uso de memoria RAM
- Cantidad de información que ha entrado y ha sido enviada a través de la red
- Número de operaciones de lectura y escritura en disco
- Datos relativos al gasto monetario en los proveedores de infraestructura

Para poder obtener esta información se han utilizado librerías que ofrecen los proveedores de servicios como Amazon AWS o Microsoft Azure. Debido a que cada proveedor de servicios proporciona datos diferentes y con formatos completamente distintos, se ha configurado una interfaz con estructura de Middleware Access Driver (MAD) [6] como la del proyecto GridWay [22], que se basa en la creación de adaptadores que habilitan el acceso a diferentes infraestructuras de producción para unificarlas bajo una misma interfaz que las federa, transformando y procesando todos los datos obtenidos para que sean consistentes y tengan el mismo formato de salida para todos los proveedores siguiendo un esquema regular.

Para seguir el esquema middleware de Gridway se ha creado una interfaz que proporciona acceso a determinados atributos modificados para ser consistentes y abstrayendo al módulo que lo utilice de ser conocedor de la estructura de los datos en crudo que proporcionan los sistemas de monitorización, en este caso las APIs de los proveedores.

Tomando como referencia un ejemplo real, se puede observar que para proporcionar la información de cantidad de operaciones en disco (DiskWriteOps) para dos instancias de

distintos proveedores, los servicios de monitorización de Amazon AWS y Microsoft Azure devuelven los siguientes valores.

- En el caso de Amazon AWS, y su monitor Cloudwatch, la información es proporcionada con el formato de recuento de la cantidad de operaciones de escritura en disco durante el periodo de tiempo especificado.

  Por ejemplo: 937421 operations

- En el caso del monitor de Azure, la información es proporcionada como la media del número de operaciones de escritura en disco por segundo en el periodo especificado.

  Por ejemplo: 7500 IOPS (operaciones de entrada/salida por segundo)

Se puede comprobar que ambos valores son inconsistentes y, por ello, es necesario procesar estos datos para que tengan el mismo formato. Utilizando la estructura MAD de GridWay podemos obtener las siguientes transformaciones para que ambas mantengan el mismo formato. En este caso el formato elegido es IOPS, así que, por un lado:

- En el caso de Microsoft Azure el valor de salida se mantiene igual, debido a que ya tiene el formato buscado.

  Resultado: 7500 IOPS

- En el caso de Amazon AWS, el formato es transformado a IOPS dividiendo el valor de las operaciones entre el periodo de tiempo en segundos durante el que se han ejecutado.

  Resultado: 7811,94 IOPS

Gracias a los resultados obtenidos tras procesar los valores iniciales, el módulo Brain no necesita conocer cómo funciona el proveedor específico de cada valor.

Sensores software

En esta parte del módulo, permite la inclusión en los servicios de actuadores que se activan a través del propio software. Para ello, se debe incluir una librería en el proyecto con el que se esté utilizando Cloud PIE y, con una pequeña configuración, el sistema adquiere la habilidad de ejecutar reglas basadas en el estado de parámetros de los servicios del proyecto.

Esta librería habilita un pequeño servicio de consulta que permite a Cloud PIE acceder a los sensores que informan del estado en el que se encuentran los atributos que han sido especificados en la configuración.

De esta manera, cuando luego se procese la información de todos los sensores, se podrá tener en cuenta valores del software, como pueden ser:

- Número de peticiones recibidas en una llamada al sistema específica. Por ejemplo, subida de imágenes a un servidor.
- Cantidad de usuarios en cola de espera.
- Valor específico de un atributo o funcionalidad. Por ejemplo, porcentaje de procesamiento de operaciones sobre un vídeo.
- Tiempo restante o consumido para la ejecución de una nueva funcionalidad.

Por tanto, el módulo Gatherer se encarga de recolectar la información, a partir de estas fuentes software y hardware, para que esté disponible para el resto de módulos del sistema Cloud PIE y que éste pueda hacer uso de ella y actúe en consecuencia.

3.2.2 Thinker

El módulo de pensamiento o razonamiento es el que se encarga de gestionar los datos obtenidos a través del módulo Gatherer y utilizarlos para tomar decisiones en relación a las transformaciones que se deben realizar sobre el ecosistema para mejorar la disponibilidad.

En este módulo, se toman como referencia los valores administrados en tiempo real sobre el estado actual del hardware y el software. Sin embargo, gracias al módulo Learner (sección 3.2.5), también se pueden tener en cuenta patrones existentes en la ejecución de los servicios.

Este servicio utiliza como base un motor de reglas llamado Wongi::Engine que permite crear reglas basadas en el algoritmo de Rete de manera sencilla y legible gracias al uso de un lenguaje de dominio específico. El algoritmo de Rete [16] es un algoritmo de reconocimiento de patrones que permite implementar motores basados en reglas, gracias a que permite emparejar tuplas de datos, los hechos, con las reglas para determinar cuál de éstas debería ejecutarse. Este algoritmo es muy eficiente ya que mantiene en memoria emparejamientos parciales para evitar tener que volver a evaluar cada regla cada vez que se hagan cambios en el sistema.

Un lenguaje de dominio específico [17], Domain Specific Language o DSL, es un lenguaje máquina que ha sido creado especialmente para su uso en un dominio de aplicación específico. Hay muchos tipos de DSL que son utilizados en diferentes áreas, como por ejemplo:

- HTML, es utilizado como lenguaje de marcado para la creación de páginas web. Por ejemplo:

```
1  <html>
2    <body>
3      <div> Hello world </div>
4    </body>
5  </html>
```

- SQL, es un DSL usado para administrar sistemas de gestión de bases de datos relacionales. Por ejemplo:

```
1   SELECT * FROM Users;
```

Las reglas utilizadas en el módulo Thinker son definidas inicialmente por el administrador como una serie de condiciones que se van a tener que cumplir en los valores de entrada, obtenidos por los sensores software y hardware, para que se produzcan las salidas correspondientes, que se envían como transiciones sobre el sistema.

Las transiciones no son llamadas directas a funciones de transformación del ecosistema, sino que son atributos constantes, etiquetas o tags, que definen una acción que debe ser realizada por otro módulo, en este caso el módulo Builder. Por ejemplo, una transición `scale-up:size:1:nginx` indica la creación de una máquina virtual para el servicio `nginx`. Estas transiciones siguen un esquema predefinido que debe ser utilizado por el administrador a la hora de crear las reglas para especificar las transformaciones deseadas en el sistema.

El esquema predefinido de las etiquetas de transición sigue el siguiente patrón:

```
operation[:op-value]?:target-service
```

A continuación se detallan los componentes de las etiquetas de transición:

operation Permite especificar la operación de transición elegida que se realizará sobre el servicio. Actualmente existen las siguientes opciones:

scale-up Permite realizar una operación de escalado horizontal que creará una máquina virtual nueva para el servicio especificado.

scale-down Permite realizar una operación de escalado horizontal que eliminará una máquina del servicio especificado.

scale-to Provoca que el sistema realice un escalado directo para mantener el número de máquinas indicado.

op-value Es un argumento opcional que puede ser añadido como valor de la operación indicada como operation. Por ejemplo: 1.

target-service Especifica el nombre del servicio sobre el que se quiere realizar una operación de transición y que ha sido definido previamente en la configuración de inicio.

El módulo Thinker tiene varias funciones que le permiten proporcionar el servicio para el que ha sido diseñado. Las funciones son las siguientes:

learn La función de learn o aprender permite al módulo aprender una lista de reglas con las que va a tomar las decisiones. Este proceso se puede realizar en cualquier momento, así que el sistema es capaz de aprender dinámicamente, favoreciendo la evolución en tiempo real al no tener que reiniciar el sistema cada vez que se quieran añadir nuevas reglas.

forget La función de forget u olvidar permite al módulo olvidar una lista de reglas que ya no resulten interesantes para el sistema. Al igual que el método de aprendizaje, la posibilidad de olvidar reglas en cualquier momento facilita la capacidad evolutiva del sistema.

decide El método de decide o decidir permite que, mediante una lista de hechos o valores de entrada, el módulo de razonamiento procese toda la información para proporcionar las transiciones que se deben ejecutar a posteriori para mejorar o escalar el entorno.

clear_mind Este método, clear mind o limpiar la mente, permite borrar los hechos aprendidos para utilizar las reglas que se obtienen a través del método decide. Es importante el uso de esta función para evitar utilizar hechos pasados a la hora de tomar nuevas decisiones.

list Permite listar las reglas y estados que hay actualmente guardados en el motor de reglas para su visualización.

En definitiva, el módulo Thinker es una parte esencial de Cloud PIE porque es el que procesa la información para tomar las decisiones sobre las transformaciones del sistema.

3.2.3 Builder

El módulo de construcción o transformación es capaz de realizar todas las funciones necesarias para que la infraestructura mute y escale cuando el módulo Brain se lo indique. Tras haber obtenido datos analíticos del módulo Gatherer y posteriormente ser procesados por el módulo Thinker, y obtener las etiquetas de transición correspondientes, este módulo es el encargado de llevar a cabo dichas transformaciones.

Para realizar las mutaciones sobre el ecosistema se utilizan las librerías que ofrecen los proveedores de infraestructura, como Amazon AWS y Microsoft Azure, y que permiten realizar, entre otras, las siguientes acciones:

- Construir y destruir máquinas virtuales
- Crear contenedores
- Gestionar redes privadas
- Provisionar volúmenes o espacios de almacenamiento

Dado que las decisiones las toma el sistema, el usuario debe crear inicialmente unas plantillas con datos de las máquinas virtuales que quiere que estén disponibles indicando información como el tipo de máquina, la localización, la capacidad, etc. Estas plantillas son creadas en un archivo de configuración en formato YAML [5], un formato de serialización de datos de alta legibilidad y que está basado en otros lenguajes de marcado como XML [8] y JSON [14]. La facilidad de lectura y escritura de este formato permite que la configuración sea muy simple y reduce el período de aprendizaje del sistema.

La estructura del archivo de inicialización que contiene las plantillas se muestra a continuación:

```
1   version: 1
2   machines:
3     machine-name-1:
4       provider: Amazon
5       type: t2.micro
6       regions:
7         - eu-west-1
8         - eu-west-2
9       script: 'echo \"hello world\"'
10
11
12    machine-name-N:
13      ...
14
15  services:
16    service-name-1:
17      machines:
18        - machine-name-1
19        - machine-name-3
20      spec_file: service1-deployment.yml
21
22   service-name-N:
23      ...
24
25  rules:
26    rule-name-1:
27      rule_file: rule1.n3
```

```
28
29      rule-name-N:
30        ...
```

En este archivo se pueden encontrar los siguientes valores:

version Indica la versión del archivo de configuración. Actualmente sólo existe la 1.

machines Especifica una lista de plantillas de máquinas virtuales. Cada plantilla se identifica por un nombre y contiene los siguientes elementos:

provider Información que identifica el proveedor de infraestructuras cloud.

type Tipo de máquina según el proveedor.

regions Lista de regiones del proveedor donde puede lanzarse la máquina.

script Script que será cargado tras inicializar la máquina virtual.

services Especifica una lista de plantillas de servicios. Cada servicio se identifica con un nombre y contiene los siguientes elementos:

machines Especifica una lista de máquinas en las que puede crearse el servicio.

spec_file Localización del archivo de configuración de Kubernetes que define los parámetros del servicio.

rules Especifica una lista de plantillas de reglas. Cada regla se identifica con un nombre y contiene los siguientes elementos:

rule_file Localización del archivo que contiene la descripción de la regla.

En la versión actual de Cloud PIE se pueden incluir scripts que serán ejecutados tras la instanciación de las máquinas y permiten realizar configuración sobre ellas, como por ejemplo: instalación de gestores de contenedores, conexión a redes privadas, actualización del

sistema, etc. Sin embargo, este sistema ha resultado ser poco eficiente, ya que para ejecutar una configuración correcta en un sistema complejo hace falta utilizar muchos comandos y la especificación sobre archivos YAML provoca muchos problemas y errores a la hora de ejecutar ciertas líneas de código. Se plantea la creación de un módulo específico para configuración y la utilización de ciertas herramientas especializadas que facilitan la configuración de máquinas virtuales en el apartado de trabajo futuro del capítulo 5 .

Por tanto, el módulo Builder se encarga de realizar las transformaciones sobre el ecosistema, proporcionando una gran versatilidad gracias a su capacidad de trabajo con múltiples proveedores y facilidad de configuración mediante plantillas.

3.2.4 Retainer

El módulo de persistencia o almacenamiento, es el encargado de guardar todos los datos que obtiene el sistema para poder ser utilizados posteriormente, así como los necesarios para mantener la consistencia del sistema en caso de fallos. Este módulo utiliza dos bases de datos y separa en ellas, por un lado, la información necesaria para ser utilizada al momento, o memoria a corto plazo, y la información que permanece guardada como histórico para aprender sobre ella y mejorar el sistema, la memoria a largo plazo, como se puede observar en la figura 3.4.

Memoria a corto plazo

Para la memoria a corto plazo se utiliza una base de datos Redis (sección 3.3.6), una base de datos rápida en la que se guarda el estado actual del entorno, los datos de inicio y los eventos pendientes de ejecutar. De esta manera, se puede visualizar en cualquier momento el estado actual del sistema y actuar en consecuencia. Si, en un momento dado, el servicio deja de funcionar y se reinicia la máquina, el sistema será capaz de recuperar la información

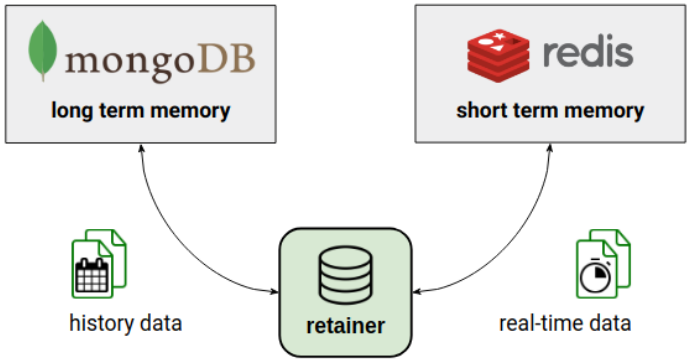

Figura 3.4: *Esquema del módulo Retainer de Cloud PIE*

del estado en el que se encontraba antes del fallo y proceder a ejecutarlo de nuevo y, así, proseguir con los eventos que estaban pendientes. Se ha decidido utilizar Redis ya que es una base de datos en memoria RAM que permite persistencia, de manera que se puede trabajar con los datos con rapidez a la vez que con la seguridad de que estos no se perderán.

Memoria a largo plazo

La memoria a largo plazo funciona con una base de datos MongoDB (sección 3.3.5), y en ella se guardan los datos que se obtienen a partir de la monitorización y las decisiones tomadas por el sistema en consecuencia; todo con información adicional como la fecha y hora a la que han sido realizadas estas acciones, y el estado en el que se encontraba el sistema a la hora de hacerlas. Se ha decidido utilizar una base de datos NoSQL [29] debido a la mutabilidad de los datos incluidos, ya que este tipo de bases de datos, y en especial MongoDB, permite guardar grandes cantidades de documentos con atributos dinámicos y trabajar sobre ellos de manera eficiente. De esta manera, si en algún momento cambian las condiciones de los datos guardados, el sistema será capaz de continuar funcionando sin realizar migraciones sobre la base de datos.

Como se ha mencionado anteriormente, el módulo Retainer hace la función de almacenaje de la información que utiliza Cloud PIE para funcionar, permitiendo su acceso en cualquier momento. Se guardan, por un lado, la información que será utilizada para el proceso de aprendizaje, y por otro, los datos del estado actual del sistema.

3.2.5 Learner

El módulo de aprendizaje o entrenamiento es el encargado de obtener patrones de comportamiento en el sistema y evolucionar las reacciones para hacer que el sistema sea más eficiente. Este worker utiliza y procesa los datos guardados en la memoria a largo plazo del módulo Retainer para generar nuevas reglas que permitan tener en cuenta los eventos que se repiten en el tiempo para ejecutar previamente soluciones que reduzcan el retraso a la hora de mutar el sistema.

Para el proceso de aprendizaje automático se ha planteado, como trabajo futuro (capítulo 5), la utilización de un sistema de reglas de asociación [25] que permite descubrir relaciones entre variables en bases de datos de gran dimensión. Su objetivo es identificar reglas sólidas de interés que son descubiertas en la base de datos aplicando ciertos análisis estadísticos. Este tipo de aprendizaje máquina se utiliza en entornos como minería web, bioinformática, detección de intrusos, etc.

El módulo Learner dispone de varias funciones que permiten al sistema aprender y generar modelos de entrenamiento:

train Este método permite entrenar los nuevos datos que se han ido obteniendo en los módulos Gatherer y Thinker para que el sistema de entrenamiento se actualice el modelo que utiliza para generar resultados.

evolve Este método permite generar nuevas reglas gracias al descubrimiento de relaciones

entre los datos obtenidos y guardados en la base de datos. Estas reglas son generadas para que posteriormente sean asimiladas por el módulo Thinker y tenidas en cuenta en el proceso de decisión.

En resumen, el módulo Learner hace la función de aprendizaje inteligente para la generación de nuevas reglas mediante técnicas de machine learning. De esta forma el sistema será capaz de evolucionar y tomar medidas de una manera más eficiente.

3.2.6 Brain

El módulo maestro o cerebro es el módulo principal de Cloud PIE que hace las funciones de controlador para el resto de módulos. Es el encargado de inicializar el sistema, de ejecutar continuamente peticiones sobre los demás servicios del proyecto y de relacionarlos entre sí.

Este módulo realiza varias tareas esenciales, entre las que se encuentran:

Inicialización del sistema

Este proceso es lo primero que se hace tras ejecutar los servicios. Permite que el sistema se inicialice correctamente con la información aportada por el usuario en las plantillas definidas anteriormente. Se compone de las siguientes etapas:

1. **Lectura de plantilla de información**
2. **Comprobación de caída (ejecuta fallback)**
3. **Persistencia de la información**
4. **Inicialización y carga de reglas**
5. **Ejecución del ciclo de decisión**

Ciclo de decisión

Este proceso se ejecuta continuamente en intervalos de tiempo definidos por el usuario en la plantilla de inicio. Es el proceso principal que se comunica continuamente con todos los demás microservicios que forman parte de Cloud PIE. Este proceso realiza la toma de decisiones sobre la evolución del sistema en función de la información recibida a través de la monitorización. Se compone de las siguientes etapas:

1. **Obtención de datos - módulo Gatherer**

   En esta etapa se obtiene la información de los servicios, a través de los sensores hardware y software que han sido implementados mediante el uso de proveedores de infraestructura y la librería de sensores software. Con ello se puede visualizar el estado actual del sistema con unos datos que siguen un formato específico para facilitar su posterior tratamiento.

2. **Persistencia de los datos obtenidos - módulo Retainer**

   Se procede a realizar el guardado de la información obtenida por el módulo Gatherer en la memoria a largo plazo del módulo Retainer para generar un histórico de datos sobre una línea temporal, y así permitir posteriormente su evaluación.

3. **Análisis de datos y obtención de resultados - módulo Thinker**

   En esta fase del ciclo se procesan los datos obtenidos a través del módulo Gatherer y, teniendo en cuenta la información aprendida mediante el módulo Learner, se generan resultados en forma de etiquetas de transición que definen transformaciones que deben ser realizadas sobre el ecosistema.

4. **Persistencia de los resultados - módulo Retainer**

   Se realiza el almacenamiento de los resultados generados a través del módulo Thinker

en la fase anterior. Estos resultados se guardan en la memoria a largo plazo del módulo Retainer como complemento al histórico de datos.

5. **Transformación del entorno - módulo Builder**

   Tras obtener las etiquetas de transición del módulo Thinker, el módulo Builder procesa estas etiquetas y realiza cambios sobre la infraestructura del sistema, creando y destruyendo instancias según sea necesario.

6. **Persistencia de checkpoint - módulo Retainer**

   En esta fase que tiene lugar después de procesar toda la información y haber ejecutado las transformaciones necesarias sobre el sistema, se guarda un punto de control o checkpoint que representa el estado en el que se encuentra el entorno en ese momento específico. Este checkpoint se guarda en la memoria a corto plazo del módulo Retainer, y permitirá obtener la información del último estado para realizar transformaciones o como medida de precaución para posibles fallos en la ejecución.

Ciclo de aprendizaje

Este proceso se ejecuta de forma asíncrona en intervalos de tiempo largos y que han sido especificados por el usuario en la configuración. Se trata de un proceso secundario en el que se utiliza la base de datos que ha sido guardada con el histórico de todos los estados y resultados que se han obtenido a lo largo de la ejecución de Cloud PIE. Se compone de las siguientes etapas:

1. **Carga de datos - módulo Retainer**

   En esta etapa se cargan los últimos datos que se han añadido a la base de datos de memoria a largo plazo, y que aún no han sido procesados por el sistema.

2. **Entrenamiento del modelo de aprendizaje - módulo Learner**

   Con la ejecución de esta fase se utilizan los datos que han sido cargados en la etapa anterior para entrenar al sistema y actualizar el modelo con nueva información. Esto permitirá producir resultados a partir del nuevo aprendizaje y hacer que el sistema se adapte mejor a lo que ha ocurrido desde el último aprendizaje.

3. **Generación de reglas - módulo Learner**

   En esta fase se procesa toda la información con la que se ha entrenado el modelo y se buscan relaciones entre las variables para generar nuevas reglas basadas en patrones que hagan que el sistema sea más inteligente y más eficiente.

4. **Aprendizaje de reglas - módulo Thinker**

   Las reglas que han sido generadas en el módulo Learner en la fase anterior son enviadas al módulo Thinker para que las aprenda y las integre en su motor de reglas. De esta manera, la próxima vez que ejecute el ciclo de decisión las utilizará y las etiquetas de transición que genere serán distintas.

En definitiva, el módulo Brain es el módulo principal maestro que controla al resto de componentes del sistema, ejecutando un proceso reiterativo que abarca desde la obtención de información hasta la aplicación de transiciones sobre el ecosistema.

3.3 Tecnologías

3.3.1 Ruby

Ruby[1] [26] es un lenguaje de programación orientado a objetos, dinámico y de código abierto, que ha sido diseñado por Yukihiro Matsumoto con el objetivo de hacer que la

[1] https://www.ruby-lang.org

programación sea agradable y rápida. Este lenguaje tiene una sintaxis muy fácil de entender y de escribir, y dispone de una amplia colección de librerías que facilitan la programación de multitud de sistemas, desde procesadores de texto y scripts, hasta entornos a gran escala.

El intérprete de código de Ruby es fácilmente instalable en los sistemas Linux más populares, y tiene una gran comunidad de desarrolladores que trabajan constantemente en mantener y mejorar las herramientas de la plataforma. Además, la mayoría de proveedores de infraestructura disponen de librerías para utilizar sus API directamente en Ruby.

Estas ventajas han hecho que, tras valorar los pros y contras y realizar algunas pruebas en comparación con Node.js y Python, se seleccione Ruby como lenguaje base para el proyecto.

3.3.2 Wongi::Engine

Wongi::Engine[2] es una motor de reglas creado en Ruby siguiendo el algoritmo de Rete clásico. Tiene un sistema complejo de gestión de reglas que permite, mediante un Lenguaje de Dominio Específico o DSL, crear condiciones que se deben cumplir para que ocurran unas acciones específicas.

Este motor de reglas ha sido elegido en comparación con otros motores de reglas porque estaba basado en el lenguaje de programación Ruby, lo que facilita su integración en el proyecto, y porque su lenguaje específico es muy simple y entendible. Además, dispone de un sistema de carga de ficheros de reglas externos, ventaja que se añade en comparación con otros motores de reglas que no permiten hacerlo, y que requieren que la programación de las reglas sea realizada directamente sobre el código del proyecto, lo que imposibilita el dinamismo de carga de reglas a posteriori y fuerza la necesidad de programar para utilizar el proyecto.

[2] https://github.com/ulfurinn/wongi-engine

3.3.3 Amazon AWS

Amazon Web Services[3] es un proveedor de servicios de infraestructura bajo demanda basado en un modelo de pago por suscripción. Forma parte de la empresa Amazon.com, Inc., y comenzó a funcionar en 2006.

Las máquinas virtuales de Amazon AWS emulan muchos de los atributos de una computadora real. Permite contratar recursos como CPUs y GPUs, memoria RAM, almacenamiento en discos duros, redes y una gran selección de sistemas operativos y de paquetes de aplicaciones software, como servicios web, bases de datos, etc.

Todos los servicios de Amazon disponen de SLAs sobre la disponibilidad, redundancia y seguridad, además de otras opciones.

La infraestructura de AWS está distribuida en granjas de servidores alrededor de todo el mundo, con distintos precios según la localización. Amazon AWS tiene centros de datos en 6 regiones de América del Norte y en otros lugares como Europa, Brasil, Singapur, Japón y Australia; con los que proporciona servicio a más de 190 países de todo el mundo.

AWS dispone de más de 90 servicios en la nube, entre los que se incluyen computación (Amazon EC2), almacenamiento (Amazon S3), redes, bases de datos, analíticas, monitorización, servicios de aplicación y de Internet of Things. Amazon proporciona acceso a estos servicios a través de sus APIs para desarrolladores, gracias a las cuales se han podido integrar varios de ellos en Cloud PIE.

[3] https://aws.amazon.com

3.3.4 Microsoft Azure

Azure[4] es un servicio de computación en la nube creado por Microsoft, similar a Amazon AWS, que permite crear recursos de infraestructura para desplegar aplicaciones y servicios en cloud. Fue lanzado en 2010 como Windows Azure y renombrado a Microsoft Azure en 2014.

Microsoft Azure soporta muchos tipos de herramientas, ya sean de Microsoft o de terceros. Al igual que AWS, dispone de SLAs para todos sus servicios en referencia a la disponibilidad, seguridad, etc.

Azure no es tan popular como Amazon AWS, pero proporciona más de 600 servicios de computación, almacenamiento, gestión de datos, mensajería, servicios de aplicaciones, etc.; a través de varios centros de datos, disponibles globalmente, que son gestionados por Microsoft. Actualmente se encuentra en más de 50 regiones alrededor del mundo, con intención de expandirse a otras nuevas.

Azure proporciona acceso a sus servicios para desarrolladores a través de sus APIs, y esto ha permitido su integración en el proyecto. Sin embargo, la gestión de sus servicios es mucho más complicada que con Amazon AWS, creando dificultades en las pruebas.

Gracias a la integración de Azure, se ha podido comprobar que el proyecto es multiplataforma y que es capaz de utilizar varios proveedores de servicios.

3.3.5 MongoDB

MongoDB[5] [12, 13] es la base de datos NoSQL por excelencia y la más utilizada hoy en día. Ha sido creada pensando en la escalabilidad y la alta disponibilidad, permitiendo

[4] https://azure.microsoft.com
[5] https://www.mongodb.com/

establecer arquitecturas multi-host complejas con datos distribuidos. Ofrece la replicación de forma nativa y es altamente tolerante a fallos, por lo que es utilizada especialmente como una solución para entornos empresariales que requieren una alta flexibilidad. Además, guarda sus datos en documentos JSON estructurados y dinámicos, lo cual permite establecer modelos no relacionales y con disparidad estructural.

Mongo ofrece la utilización de gran cantidad de tipos de atributo que permiten guardar múltiples formatos de información, incluyendo algunos elementos complejos como arrays, diccionarios y modelos anidados. Posteriormente, estos campos pueden ser buscados en la base de datos de forma directa, mediante uso de expresiones regulares, o incluso con búsquedas más complejas como Agregación y Map-Reduce.

Cualquier campo de la base de datos puede ser indexado, favoreciendo la velocidad en búsquedas ordenadas y agrupadas.

Agregación

La agregación [3] es un proceso mediante el cual Mongo procesa todos los registros de datos y devuelve resultados según la consulta especificada. Mongo agrupa los datos y les aplica las operaciones en conjunto para ofrecer un único resultado final.

Map-Reduce

Map-Reduce [15] es un modelo más costoso de procesamiento de los datos para obtener resultados basados en la ejecución de funciones distribuidas sobre los datos. Se basa en una primera etapa de Map, en la cual agrupa los datos según características afines, y una segunda etapa de Reduce, en la que aplica operaciones sobre los datos para obtener el resultado final. Normalmente se utiliza de forma distribuida para acelerar cálculos de búsqueda muy complejos. Se puede ver una operación de map-reduce en MongoDB en la figura 3.5.

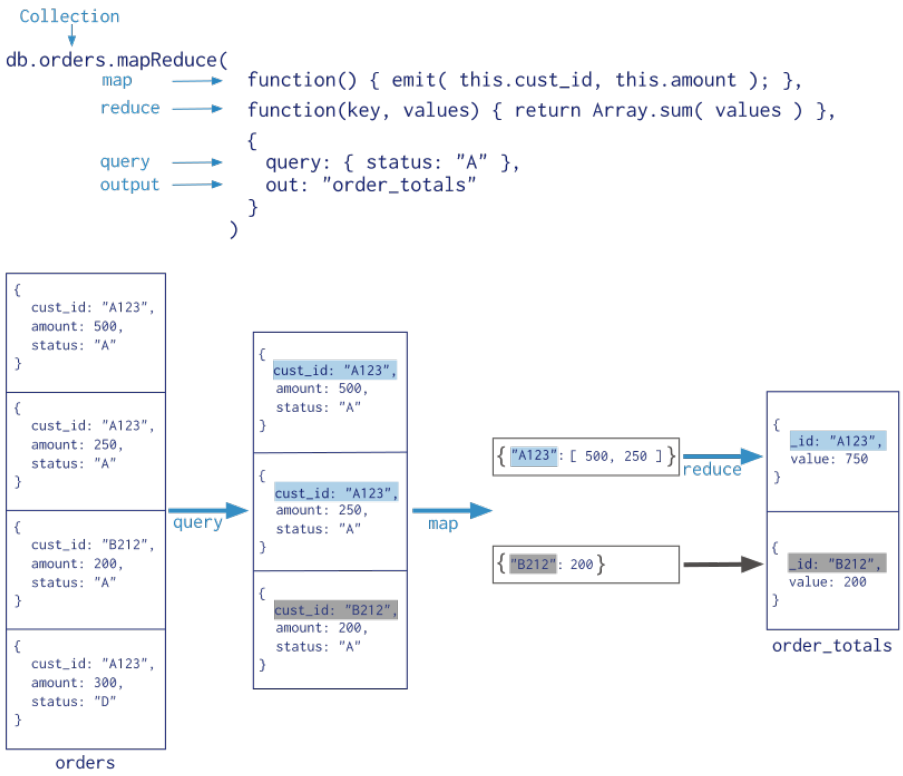

Figura 3.5: *Operación map-reduce en MongoDB*

Además, MongoDB permite establecer un balanceador de carga para poder replicar los datos y distribuirlos mediante sharding. El balanceador hace las veces de enrutador y establece los parámetros según las peticiones y el nivel de carga en cada nodo.

La replicación de Mongo es muy similar a la de otros sistemas como PostgreSQL, está orientada a mantener backups distribuidos para evitar la pérdida de datos, pero también pueden ser utilizados como nodos de lectura. Además MongoDB mantiene un sistema de votaciones entre los nodos para que estos decidan aplicando un sistema democrático quién es el nodo master de la replicación.

El sharding es una de las grandes ventajas que proporciona MongoDB, debido a que permite distribuir la base de datos en shards o fragmentos separados en distintos nodos, y realizar operaciones de lectura distribuidas, encargando a cada nodo el procesamiento de los

datos que pertenecen a su fragmento. Para ello, MongoDB separa los datos según el campo del modelo establecido en la configuración, por ejemplo la id o el nombre, y posteriormente distribuye las peticiones para repartir el trabajo; finalmente el router se encarga de unir esta información y devolverla como si nunca hubiera estado separada.

MongoDB ha sido elegido como base de datos para guardar toda la información permanente obtenida a través de la monitorización y la toma de decisiones, y que será necesaria para el aprendizaje máquina y para generar resultados que permitan evolucionar el sistema. La gran versatilidad de MongoDB y su eficiencia al trabajar con grandes cantidades de datos son algunos de los factores que han hecho que sea elegida como base de datos para la memoria a largo plazo del módulo Retainer.

3.3.6 Redis

Redis[6] [10, 13] es un proyecto de base de datos creado por Salvatore Sanfilippo que almacena la información en memoria en tablas clave/valor, pero que también puede persistir los datos en disco. Soporta numerosas estructuras de datos como listas, mapas, sets, etc. Existen varios sistemas mediante los que Redis guarda los datos en disco para evitar posibles pérdidas ante un fallo general, aunque, por defecto Redis ya guarda la información cada 2 segundos en disco como medida preventiva.

Redis soporta replicación y clustering, y es muy eficiente cuando los datos no necesitan permanecer guardados de manera continua, ya que es su principal objetivo, mantenerlos de manera temporal y desecharlos cuando sea necesario.

Esto ha hecho que sea muy útil como base de datos para la memoria a corto plazo del módulo Retainer. Ya que los elementos que se guardan en este parte del módulo son datos perecederos sobre el estado actual del sistema, en forma de checkpoints, y las acciones que

[6] https://redis.io

deben ser realizadas para mutar el entorno. Sin embargo, estos datos sólo sirven en tiempo real y no se van a utilizar a largo plazo, así que esta base de datos es idónea para trabajar con velocidad y datos que no deben permanecer guardados de manera constante.

3.3.7 Docker

Docker[7] [28, 13] es la plataforma líder de contenedores software. Fue creada en 2013 por Solomon Hykes como parte de un proyecto interno de dotCloud, una empresa que se dedicaba a ofrecer su Plataforma como Servicio y que finalmente se transformó en Docker, Inc., la empresa que está detrás del proyecto. Es de código libre y es utilizada por desarrolladores para eliminar los problemas que conlleva la colaboración en proyectos y que suelen estar relacionados con dependencias de versionado y librerías. Sin embargo, su principal funcionalidad es que permite ejecutar y configurar grupos de aplicaciones aislándolas en contenedores para obtener una mejor capacidad de cómputo y un mejor aprovechamiento de los recursos. Es imprescindible para las empresas que crean software para establecer metodologías ágiles de despliegue y abastecer con nuevas mejoras a sus sistemas con mayor celeridad, seguridad y fiabilidad en sistemas Linux y Windows

Los contenedores son paquetes ligeros y aislados que contienen software ejecutable y que incluyen todo lo que se necesita para utilizarlos, ya sea código, librerías, configuración y otras herramientas. Siguen un estándar abierto que les permite ejecutarse siempre de la misma manera, independientemente del entorno donde sean utilizados, ya sea Linux o Windows, máquinas virtuales o servidores dedicados. Los contenedores aíslan el software del resto de elementos y proporcionan un sistema fiable a la hora de crear entornos de desarrollo y de producción.

Se pueden ejecutar múltiples contenedores en la misma máquina y compartir el kernel del

[7] https://www.docker.com

Sistema Operativo con otros contenedores (figura 3.6), cada uno ejecutándose en un proceso y un espacio completamente aislado. Pesan mucho menos que las máquinas virtuales y se ejecutan de manera casi inmediata.

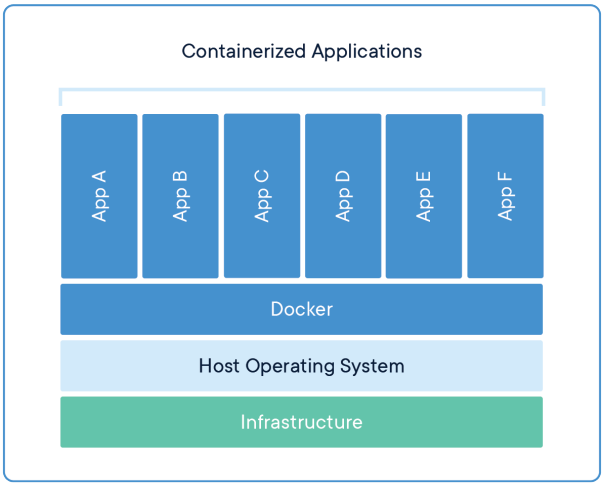

Figura 3.6: *Estructura de un Contenedor*

Un contenedor puede funcionar sin problemas siendo ejecutado sobre una máquina virtual. Cada máquina virtual puede tener su propio sistema de contenedores y, a su vez, ejecutar contenedores por separado sin ningún tipo de relación, como se puede observar en la figura 3.7.

Tal ha sido la repercusión de la tecnología creada por Docker, que ha cambiado completamente el paradigma de la industria software y cloud. En 2017 alcanzó la cifra de 12 mil millones de descargas, y todas las empresas que proporcionan Infraestructura como Servicio y Plataformas como Servicio la utilizan, incluso han surgido muchas empresas en torno a la misma tecnología.

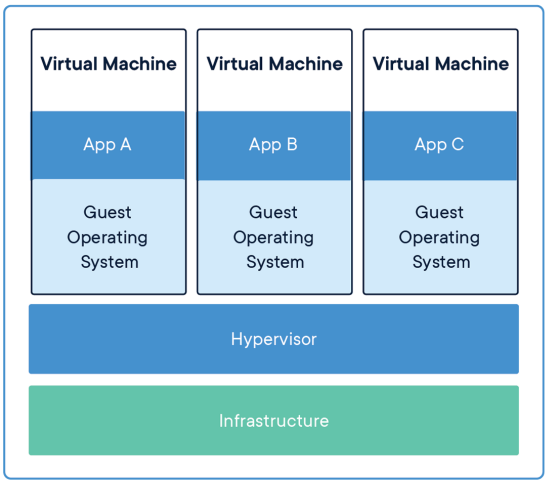

Figura 3.7: *Contenedores sobre Máquinas Virtuales*

Docker distingue su uso en tres ámbitos principales para los que proporciona ventajas diferenciadoras:

- Desarrolladores
  - Desarrollo, testing y despliegue de aplicaciones programadas en cualquier lenguaje sin riesgo de incompatibilidades.
  - Creación de contenedores específicos para no tener que reinstalar y reconfigurar el software, reduciendo el tiempo de trabajo.
  - Simulación de entornos de producción para testing.
- Operadores
  - Mayor velocidad a la hora de desplegar software.
  - Escalado rápido de miles de nodos y contenedores, para satisfacer los picos en la demanda de consumo.
  - Eficiencia a la hora de resolver problemas y conflictos de mantenimiento.
  - Distribución y compartición de contenedores en Registros Docker.

  - Facilidad a la hora de compartir aplicaciones.
  - Mayor seguridad

- Negocios
  - Framework unificado para todas las aplicaciones.
  - Innovar con velocidad y escalar.
  - Colaboración entre desarrolladores y operadores

Uno de los elementos más interesantes de Docker es que permite utilizar distintos proveedores de infraestructuras y plataformas como servicio, crear clústeres unificándolos y abstraer la lógica de infraestructuras.

Docker ha sido elegido como tecnología de contenerización para la configuración de las máquinas virtuales. Pese a que no se ha podido incluir el módulo de configuración en el proyecto, se ha realizado una investigación sobre este sistema y pruebas, y se ha planteado su utilización en la fase de trabajo futuro.

3.3.8 Kubernetes

Kubernetes [8] [21, 13] es una plataforma open-source para automatizar el despliegue, escalado y operaciones de contenedores sobre clústeres, y manteniendo una estructura centralizada. Es un proyecto que fue creado por Google en 2014 y que, con la colaboración de la comunidad, ha evolucionado hasta ser una alternativa muy viable a la hora de utilizar este tipo de tecnología. Google Cloud la utiliza por defecto como sistema para desplegar en su PaaS, App Engine.

Kubernetes sigue un modelo más complejo, pero también más completo, que Docker

[8] https://kubernetes.io

Swarm. Es necesario un nivel avanzado en tratamiento con contenedores y un alto conocimiento del funcionamiento general para utilizarlo, y la curva de aprendizaje es, también, alta. Sin embargo, su gran versatilidad permite realizar configuraciones eficientes y robustas, y su funcionamiento es más fácil de mantener a largo plazo que otros gestores de contenedores como Docker Swarm.

Kubernetes ha sido elegido como tecnología de gestión de contenedores para la configuración de las máquinas virtuales. Pese a que no se ha podido incluir el módulo de configuración en el proyecto, se ha realizado una investigación sobre este sistema y pruebas, y se ha planteado su utilización en la fase de trabajo futuro.

Capítulo 4

Experimentación

4.1 Casos de uso

Desde las fases más tempranas de creación de Cloud PIE, se han ideado ejemplos de proyectos donde podría utilizarse la plataforma con el objetivo de fortalecer la validez de la funcionalidad planteada. Estos casos de uso no sólo han servido para verificar que existen muchos entornos donde sería una herramienta útil, sino que también han ayudado a acercar el proyecto a una idea cada vez más completa y más cercana a las necesidades reales.

Hay un amplio espectro de posibilidades de casos de uso donde se podría aplicar Cloud PIE, pero vamos a centrarnos en tres modelos que van a ayudar, por un lado, a entender mejor el proyecto; y, por otro, como ayuda de cara a las pruebas y los experimentos.

A continuación se presentan tres modelos de casos de uso donde podría ser útil usar Cloud PIE para gestionar la infraestructura.

4.1.1 Massive Multiplayer Videogame

Este caso de uso consiste en un videojuego multijugador masivo, como el de la figura 4.1 en el que hay una cantidad elevada de jugadores de forma constante. El videojuego se compone de una parte servidor, donde se manejan los algoritmos del juego, y una parte cliente, donde los jugadores pueden visualizar e interactuar con los elementos gráficos del mundo virtual.

El videojuego une a grupos de 100 jugadores en una partida en la que, tras comenzar, pueden interactuar entre sí mediante la dinámica del juego.

Figura 4.1: *Captura de Fortnite, un ejemplo de videojuego multijugador masivo*

La parte cliente, creada con un motor de videojuegos, se ejecuta directamente sobre el dispositivo de los usuarios que juegan y realiza peticiones al servidor del videojuego según las decisiones que toma el usuario.

La parte servidor se ejecuta en un entorno cloud, formado por un clúster que dispone de varios servicios montados sobre instancias.

Los servicios que componen la parte de servidor son:

- Balanceador de carga a donde llegan todas las peticiones del cliente y que las reparte entre los distintos servicios replicados.

- Servicio de menús donde el usuario puede interactuar con el sistema y obtener información de rankings, perfil personal y otros datos informativos.

- Servicio de gestión de colas y emparejamiento, ya que los usuarios necesitan entrar en cola para unirse a una partida. Todos los jugadores interactúan con este servicio durante el tiempo de espera y hasta que la partida comienza.

- Servicio de partida, con el que los jugadores se comunican directamente cuando pasan a formar parte de la partida. Este servicio conecta con todos los jugadores mediante sockets para ayudar a acelerar la comunicación y hacer que el juego tenga fluidez a la hora de distribuir las actualizaciones del entorno virtual a todos los jugadores.

Existen más elementos que forman parte del servidor, como bases de datos, redes privadas, firewalls, etc.

Cloud PIE puede ser realmente útil en este proyecto. No sólo por ser capaz de gestionar la carga en caso de que exista un pico de usuarios a la hora de proporcionar los servicios, sino porque es capaz de aprender de ese pico de usuarios y localizar los momentos temporales donde hay más jugadores para que el sistema se adapte antes de saturarse. Es decir, prever el escalamiento antes de que haya un pico de usuarios.

Un elemento en el que este proyecto destaca especialmente para su uso con Cloud PIE es la posibilidad de generar servicios de partida, entendiendo como tal la creación de instancias dedicadas a una partida de juego, mediante el uso de sensores software, cuando el sistema se acerca a obtener la cantidad de 100 usuarios requeridos para que la partida pueda comenzar.

Esto permite a los administradores de sistemas del juego la posibilidad de no tener que mantener constantemente máquinas levantadas con el servicio, y escalar previamente

al lanzamiento de una partida con el tiempo suficiente para que no sea percibido por el jugador.

Tradicionalmente, los juegos con esta problemática mantienen una serie fija de servidores levantados que se encargan de gestionar las partidas basándose en datos que han obtenido tras visualizar los picos de usuarios. Igualmente mantienen un sistema de reglas que les permite escalar en caso de que esté habiendo una cantidad de usuarios mayor. Sin embargo, gracias a Cloud PIE, no sería necesario mantener estos servidores fijos, sino que se instanciarían en el momento adecuado, reduciendo con ello costes y optimizando los recursos.

4.1.2 Masquerade Botnet

Una botnet [23] es un conjunto de robots informáticos que se ejecutan de manera autónoma. Generalmente una botnet se ejecuta en la nube, unificando una gran cantidad de dispositivos infectados por malware para realizar algún tipo de tarea maliciosa en conjunto.

El objetivo más común para el que se suelen crear botnets son los ataques de denegación del servicio, que bloquean el acceso a un sitio web u otro tipo de servicio al recibir un ataque constante de peticiones masivas por multitud de dispositivos que saturan su red y sus sistemas.

Este caso de uso se basa en el concepto de botnet, pero sin utilizar como componentes dispositivos infectados, sino instancias en proveedores cloud. Sin embargo, también podría realizarse con ese modelo tradicional haciendo algunas adaptaciones al proyecto Cloud PIE.

Además, el objetivo de esta botnet tampoco es el mismo que los tradicionales, que lo que buscan es, por ejemplo, realizar ataques de denegación de servicio, envío de spam, como en la figura 4.2, o, en los últimos tiempos, minería de bitcoins.

El objetivo de esta botnet es crear un sistema inteligente que simule un entorno real

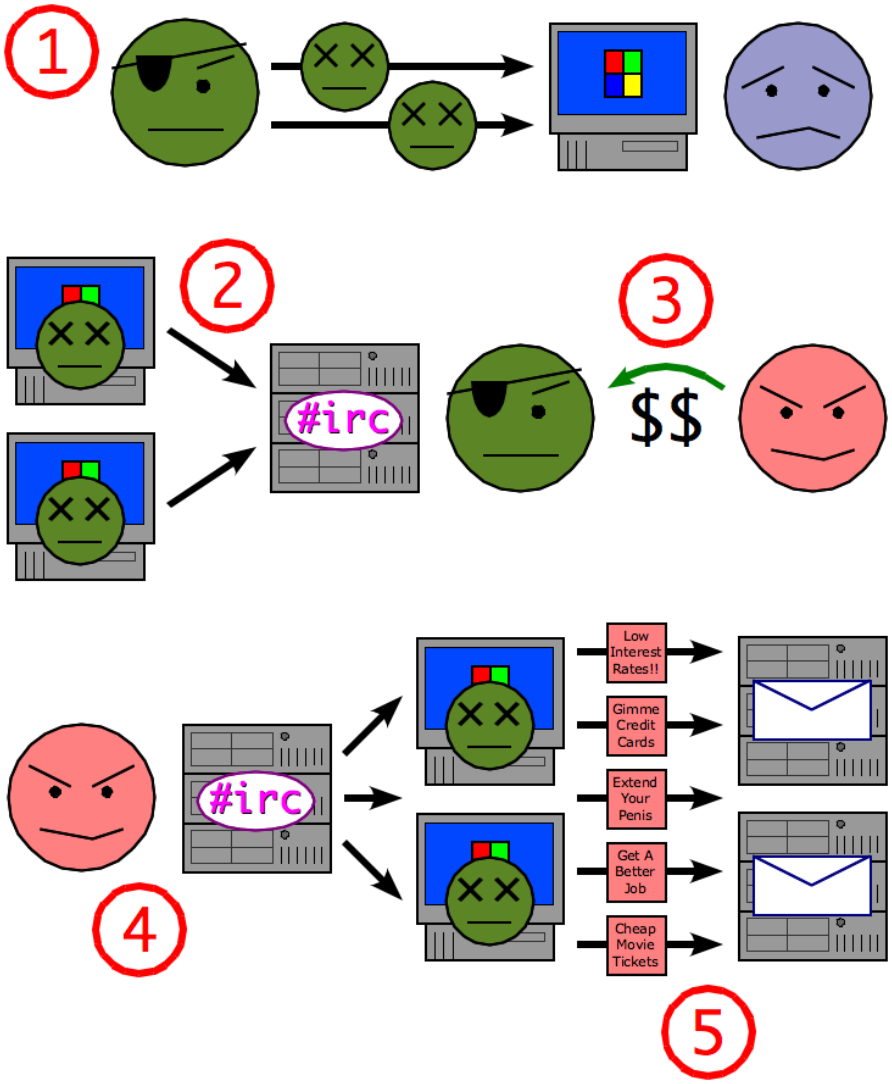

Figura 4.2: *Ejemplo de botnet usada para enviar spam*

formado por usuarios ficticios y servicios web falsos. Esta botnet estaría compuesta de servicios que utilizan modelos específicos para engañar a otros servicios haciéndoles creer que son sistemas reales y que están interactuando con la nube.

Por un lado, existen modelos de blogs, tiendas, foros, etc. que simulan sitios web con los que el usuario interactúa y que, a su vez, hacen referencias a otros de estos sitios web y a sitios web externos.

Por otro lado, existen modelos de usuario con rutinas distintas basadas en agentes inteligentes [43] que simulan estar interactuando con los sitios web falsos y con otros externos. Ejemplos de acciones que realizan son: crear posts en los blogs, postear en los foros, comprar en las tiendas, etc.

Se trata de un entorno que evoluciona constantemente por sí mismo, mutando los servicios y los usuarios con cierta periodicidad y con tiempo de caducidad para evitar ser detectado y para simular mejor un entorno real.

Obviamente, este caso de uso plantea un sistema complejo que necesita un ecosistema donde poder crecer y evolucionar en tiempo real. Aquí es donde entra la usabilidad de Cloud PIE, ya que éste le permitiría controlar la infraestructura por completo, además de facilitarle la activación de sensores software que pueden ser aprovechados por los servicios para evolucionar y propagar nuevos usuarios ficticios y webs ficticias.

4.1.3 Job Processor

Este caso de uso trata un ejemplo más específico y especializado que es la ejecución de jobs o trabajos que realiza un sistema para ejecutar tareas predefinidas y distribuidas.

El modelo sigue el caso de un sistema que tiene que estar constantemente lanzando trabajos para que sean ejecutados en máquinas por separado, aprovechando con ello al máximo los recursos y adaptando el sistema para favorecer la ejecución de los trabajos que tengan más prioridad.

En este caso Cloud PIE habilita la posibilidad de mutar el sistema constantemente con las necesidades específicas que tenga en cada momento el procesamiento de jobs. En unos casos puede crecer para ejecutar un job por máquina, o incluso crear máquinas con mayores recursos que se adapten a la prioridad específica de cada job.

Por otro lado, el sistema evolutivo de Cloud PIE puede ser muy útil para detectar patrones y adaptarse mejor al sistema.

En mucho casos, los job processors ejecutan trabajos similares de manera reiterada. Algunos ejemplos de esto pueden ser un sistema de transformación de vídeos, que constantemente esté optimizando los vídeos reduciendo su calidad y realizando otras operaciones sobre ellos cuando son subidos a una plataforma web o un sistema que ejecute tareas repetitivas sobre valores de bases de datos para obtener resultados acotados.

En estos ejemplos, Cloud PIE puede ser capaz de detectar el momento idóneo para lanzar una nueva instancia, o incluso detectar los intervalos donde se lanzan más jobs y aprovisionar recursos para ello con antelación.

4.2 Experimentos

Tras haber implementado y estructurado la arquitectura del proyecto, es necesario verificar que funciona correctamente y que se cumplen los objetivos que se buscaban con el sistema que se ha planteado. Para ello se ha diseñado una fase de experimentos que permitirá probar el correcto desarrollo y funcionamiento de todos y cada uno de los módulos, así como del proyecto unificado por completo.

Primero se ha realizado una fase de experimentación con proveedores de infraestructura con el objetivo de probar algunos servicios donde podría aplicarse Cloud PIE a la hora de crear máquinas virtuales.

Luego se ha probado cada módulo de trabajo por separado para verificar el correcto funcionamiento y la autonomía de todos ellos.

Posteriormente, se han unificado todos los workers y se ha procedido a realizar pruebas

en conjunto, para comprobar que los ciclos planteados eran correctos.

Finalmente, se ha realizado una simulación de un sistema complejo, utilizando uno de los casos de uso, con el objetivo de probar si el proyecto sería útil para un sistema como el descrito, además de obtener resultados basados en lo que sería un uso real.

4.2.1 Experimentación con proveedores de infraestructura

Uno de los elementos que más problemas y dificultades ha dado es el trabajo con proveedores de servicios.

El motivo es que este proyecto está planteado con el objetivo de gestionar infraestructuras complejas y, en caso de tener que probar con muchas máquinas virtuales, el coste podría ser muy elevado. Para evitar este coste, se ha optado por crear cuentas con recursos gratuitos para estudiantes, ya que este trabajo está dentro del marco de un Trabajo de Fin de Máster y cuenta con recursos limitados.

Por lo tanto, el problema no ha sido la implementación de las librerías que proporcionan estos proveedores, sino el hecho de tener que lidiar con la creación de dichas cuentas para estudiante; puesto que no todos los proveedores disponen de este tipo de cuentas, y los que lo hacen tienen un proceso muy tedioso para poder optar a ellas.

Amazon AWS

Amazon AWS es uno de los proveedores cloud que más facilidades da a la hora de probar sus servicios y de utilizarlos de manera gratuita. Por un lado, proporciona acceso gratuito a su entorno para estudiantes con un límite de consumo en créditos de $75 para miembros de instituciones ligadas a AWS Educate[1] o de $30 para instituciones que no sean miembro.

[1] https://aws.amazon.com/es/education/awseducate

Además, Amazon tiene una alianza con Github[2], y permite a estudiantes que estén registrados en Github Education[3] adquirir un bonus en el límite de crédito que tienen en AWS Educate, pasando a disponer de un crédito de $115 y $50, respectivamente.

Para el proyecto se ha creado inicialmente una cuenta de AWS Educate con las ventajas añadidas de Github Education, y ha servido para realizar las pruebas iniciales. Sin embargo, el crédito del que se disponía no era muy elevado y se gastaba enseguida, así que las pruebas han tenido que limitarse mucho. Además, ha sido necesario destruir las máquinas con las que se estaba probando tras cada prueba, para evitar consumir los créditos de manera instantánea.

Lamentablemente, la cuenta de AWS Educate ha sufrido cambios en sus condiciones durante el tiempo en que se estaba utilizando para las pruebas, y esto ha perjudicado al proyecto. Las nuevas condiciones impiden utilizar el crédito libremente y sólo está disponible para ser utilizado con tutoriales fijos que disponen en la plataforma, así que ha habido que buscar otra solución.

Tras investigar las posibilidades, se ha descubierto que Amazon AWS también proporciona un límite de crédito gratuito durante 1 año en cuentas reales de nueva creación. Esta oferta no está ligada a AWS Educate, sino que se ofrece de forma general a los clientes.

Gracias a esta nueva cuenta se ha podido realizar todas las pruebas restantes, aunque también ha habido que limitar el espectro para evitar sobrecostes.

Microsoft Azure

Microsoft Azure dispone también de cuentas orientadas a estudiantes que permiten utilizar su infraestructura con límites gratuitos. Sin embargo, la dificultad que supone su interfaz

[2] https://github.com
[3] https://education.github.com/

(figura 4.3) para gestionar la información ha hecho que se descarte la utilización de esta cuenta para el proyecto. La impresión inicial, tras muchos intentos de configuración, es que los límites incluyen ciertas restricciones para utilizar sus APIs y utilizar las máquinas virtuales libremente.

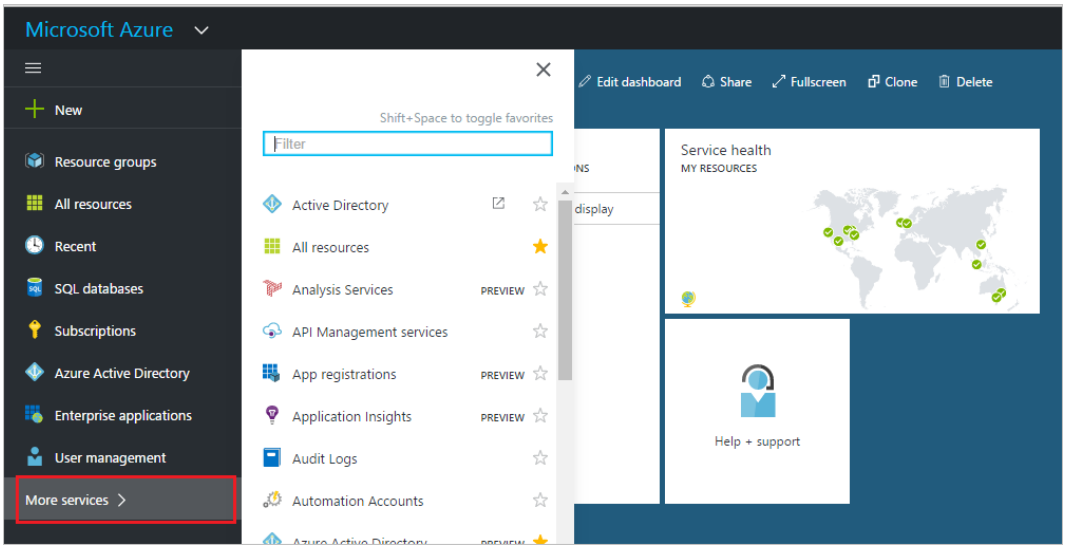

Figura 4.3: *Captura del panel de control de Microsoft Azure*

Sin embargo, se han podido realizar las pruebas correspondientes, de forma limitada, sobre Microsoft Azure, gracias a que se ha conseguido obtener acceso a una cuenta de Microsoft Azure real, sin las limitaciones de las cuentas de estudiante.

Lamentablemente, pese a la existencia de librerías en Microsoft Azure para el lenguaje de programación Ruby, la documentación proporcionada por Microsoft y la complejidad del diseño de sus librerías han impedido una implementación correcta y no se ha podido obtener el acceso deseado a sus APIs.

Otros

Existen otros proveedores de infraestructura que proporcionan créditos gratuitos a cuentas de nueva creación.

Este es el caso de Google Cloud Platform [4], que proporciona un nivel gratuito de $300 durante 12 meses para utilizar su infraestructura. Sin embargo, pese a que también dispone de créditos gratuitos para proyectos de investigación de estudiantes, la aplicación a estas ayudas es más compleja y ha sido descartada. También se ha descartado la utilización de su crédito gratuito ya que era suficiente con utilizar Amazon AWS y Microsoft Azure para comprobar que la arquitectura era multiplataforma.

DigitalOcean[5] es otro proveedor de infraestructura que también ha sido valorado para las pruebas del proyecto. Sin embargo, el crédito gratuito que habilita es muy escaso para realizar las pruebas y no se han encontrado disponibles ayudas para estudiantes ni proyectos de investigación.

4.2.2 Pruebas unitarias por módulo

Después de haber probado los proveedores de infraestructura y haber sido capaz de configurar las librerías se ha procedido a probar los módulos que componen el proyecto de Cloud PIE por separado. A continuación se explican los experimentos que se han realizado con cada módulo:

Gatherer

Se han realizado pruebas con el módulo de monitorización destinadas a comprobar el correcto funcionamiento de éste, así como verificar que el módulo era capaz de realizar las siguientes tareas:

[4] https://cloud.google.com
[5] https://www.digitalocean.com

1. **Validación de sensores hardware**

   Con estas pruebas se buscaba validar la capacidad del módulo de obtener datos de monitorización de proveedores de infraestructura, en este caso de Amazon AWS y de Microsoft Azure. Para ello se han implementado las librerías de Ruby de ambos proveedores y, posteriormente, se han realizado peticiones a sus sistemas con el objetivo de verificar que se estaban obteniendo los resultados deseados.

   Entre las operaciones que se han probado está la obtención de datos de monitorización sobre máquinas virtuales instanciadas, la capacidad de filtrar los resultados basados en el tiempo y la capacidad de agrupar los resultados por conjuntos de máquinas.

   Se han obtenido correctamente como resultados las siguientes métricas:

   - Uso de la CPU

     Valor: 0.10056497175141259

   - Uso de disco (incluyendo lectura y escritura)

     Bytes de lectura: 0.0 bytes

     Bytes de escritura: 0.0 bytes

     Operaciones de lectura: 0.0 IOPS

     Operaciones de escritura: 0.0 IOPS

   - Información sobre la red (paquetes de entrada y salida)

     Bytes entrantes: 56.0 bytes

     Bytes salientes: 28.0 bytes

     Paquetes entrantes: 1.0

     Paquetes salientes: 1.0

   - Crédito consumido

     Uso: 0.003866

     Balance: 46.580522

Además se ha conseguido filtrar estos resultados en función del tiempo. Estos intervalos de tiempo pueden ser muy reducidos, alcanzando márgenes de 5 minutos.

También se ha logrado obtener la información de monitorización tanto de una máquina de forma unitaria, como de un conjunto de máquinas definido mediante grupos, etiquetas, y múltiples opciones de filtrado.

Además, las métricas permiten la aplicación de fórmulas estadísticas para producir los resultados, como por ejemplo la media, sumatorio, el máximo, el mínimo, etc.

2. **Validación de la estructura Middleware Access Driver y abstracción de resultados**

   Uno de los elementos principales de este módulo es que sea abstracto a la hora de proporcionar los datos de monitorización y que los resultados tengan el mismo formato, independientemente del proveedor que esté utilizando en cada momento.

   Para esta estructura se ha creado una clase intermedia que conoce las diferencias de los formatos que tienen los datos según el proveedor y siempre reproduce el mismo formato de salida con cada tipo de petición.

   Para ello se han realizado pruebas solicitando valores de monitorización con distintos proveedores y se ha verificado que los resultados obtenidos siguen el mismo formato, facilitando su uso e ignorando si provienen de un proveedor o de otro.

3. **Prueba de concepto de sensores software**

   Se ha realizado una prueba de concepto para comprobar el funcionamiento de los sensores software y así plantear su implementación como trabajo futuro.

   La prueba que se ha realizado consiste en la generación de un servicio simple que simula tener instalada la librería de Cloud PIE para la utilización de sensores software.

   Este servicio responde al módulo Gatherer con un valor creciente en el tiempo que simula el número de usuarios en cola de espera.

Se ha conseguido obtener correctamente la información del sensor software a través del módulo Gatherer y se ha incluido como respuesta junto con los valores obtenidos por los sensores hardware, permitiendo su posterior uso en el motor de reglas del módulo Thinker.

Gracias a la realización de estas pruebas, se ha podido comprobar que el componente de sensores software proporciona al sistema valores útiles cuyo uso puede resultar interesante al aplicar Cloud PIE con ciertos servicios.

Estos sensores proporcionan nuevos hechos sobre los que basar la evolución del sistema y hacen que la evaluación no se limite únicamente al estado del hardware de la infraestructura.

4. **Capacidad del módulo de ser autónomo y ser utilizado externamente**

   Esta prueba se ha realizado con el objetivo de verificar que el módulo podría componerse en forma de microservicio y que el módulo principal, Brain, sea capaz de ejecutar tareas sobre él como un servicio externo.

   Con este objetivo se han realizado peticiones con el módulo Brain a este módulo y se ha verificado que funciona correctamente.

Thinker

El módulo de razonamiento es uno de los módulos esenciales ya que es el que procesa los resultados de monitorización para obtener las etiquetas de transición del entorno. Al ser un módulo muy ligado a la inteligencia artificial, se han investigado y probado varios motores de reglas para decidir cuál era el que mejor se adaptaba a las necesidades del proyecto.

Tras realizar varias pruebas simples con Wongi::Engine se ha tomado la decisión de utilizar este motor de reglas, ya que, aparte de su gran capacidad de personalización de las reglas, dispone de la posibilidad de generar resultados como valores y, también, como

ejecución de funciones. Además Wongi::Engine resulta muy útil para el proyecto al poder crear las reglas con código Ruby y, también, a partir de ficheros externos importados.

La intención de las pruebas ha sido comprobar que el módulo Thinker, junto con el motor de reglas Wongi::Engine, adquiere las siguientes capacidades:

1. **Capacidad de ejecución de reglas eficiente**

   Para ello se ha probado a ejecutar una serie de reglas creadas in situ que simulan los resultados de la monitorización, buscando obtener los resultados que se dan cuando ocurren unas situaciones específicas en un entorno real.

   Pese a la facilidad de integración del motor de reglas de Wongi::Engine, ha sido complicado configurar el módulo para ejecutar las funciones de la manera que se buscaba, ya que requería el aprendizaje de su DSL para el funcionamiento de las reglas. Además, puesto que no había documentación al respecto, se ha tenido que investigar el repositorio de código de la librería para conseguir saber cómo se utiliza la importación directa de ficheros de reglas, en vez de tener que establecer estas reglas programáticamente.

   Un ejemplo de reglas que ha sido utilizado es el siguiente:

```
1  rule('Api High CPU Usage') do
2    forall do
3      has :service, :is, :api
4      has :today, :is_not, :sunday
5      has :cpu_usage, :is, :VALUE
6      greater :VALUE, 0.9
7    end
8    make do
9      gen :result, :is, 'scale-up:1:api'
10   end
11 end
```

   En la regla indicada con el nombre `Api High CPU Usage` se puede observar que ha sido especificado que debe cumplir las siguientes condiciones:

- El nombre del servicio debe ser `api`
- El día no debe ser domingo
- El valor de uso de la cpu debe ser mayor que 0.9 (90 %)

Con estas condiciones, la regla genera el siguiente resultado:

- Etiqueta de transición `scale-up:1:api` que especifica que se debe escalar hacia arriba el servicio `api` una vez.

Finalmente, los resultados han sido favorables, ya que Wongi::Engine ha probado ser muy útil y estar bien estructurado, en comparación con otros motores de reglas.

2. **Posibilidad de importación de reglas dinámicamente**

   Uno de los elementos que se buscaban a la hora de elegir un motor de reglas, era la capacidad de éste de mutar y utilizar nuevas reglas sin tener que volver a inicializarlo. El motivo es que, al tratarse de un ecosistema dinámico con aprendizaje, las reglas sobre las que se basa pueden variar y ajustarse para ser más eficiente.

   Para ello, se ha probado a realizar aprendizaje de nuevas reglas tras haber utilizado el motor con unas reglas iniciales y el resultado ha sido satisfactorio.

3. **Capacidad del módulo de ser autónomo y ser utilizado externamente**

   Esta prueba se ha realizado con el objetivo de verificar que el módulo podría componerse en forma de microservicio y que el módulo principal, Brain, sea capaz de ejecutar tareas sobre él como un servicio externo.

   Con este objetivo se han realizado peticiones con el módulo Brain a este módulo y se ha verificado que funciona correctamente.

Builder

Las pruebas realizadas sobre el módulo de construcción han sido necesarias para verificar que era capaz de crear instancias de infraestructura basadas en los resultados del módulo Thinker. Debido a que este módulo trabaja con varios proveedores distintos, al igual que el módulo Gatherer, debe ser capaz de abstraer la gestión de las librerías y que los módulos externos simplemente le soliciten la creación de infraestructura mediante etiquetas de transición.

Las pruebas realizadas son las siguientes:

1. **Capacidad de crear y destruir infraestructura de distintos proveedores**

   Para validar que esta funcionalidad está implementada correctamente se han realizado una serie de pruebas creando múltiples instancias de máquinas virtuales con diferentes configuraciones.

   Se han creado máquinas de tipo `t2.micro`[6] de forma sencilla y se ha podido comprobar que, para que el sistema sea eficaz, no vale con simplemente crear estas máquinas, sino que hay que realizar cierta configuración sobre ellas y añadir otros módulos que proporcionan los proveedores de servicios como:

   - Grupos de seguridad
   - Etiquetas de marcado
   - Claves para su posterior acceso.

   Además, se ha podido comprobar que el tiempo de instanciación e inicialización de una máquina no es instantáneo, sino que tiene cierto retraso.

   Tras realizar pruebas creando múltiples instancias, se ha podido obtener que la creación de una máquina virtual de tipo `t2.micro` en la región de `eu-west-3c` para una imagen

[6] https://aws.amazon.com/es/ec2/instance-types

`ami-20ee5e5d` (Ubuntu) tiene un retardo medio de 4 minutos. A este retardo habría que sumarle el proceso de configuración de la máquina tras su instanciación, que no ha sido calculado al ser totalmente dependiente del tipo de servicio.

Éste es un factor a tener en cuenta y que valida la necesidad de encontrar patrones que permitan modificar la infraestructura con antelación, para evitar la falta de disponibilidad durante ese retraso.

2. **Verificar que las etiquetas de transición son funcionales**

   Las etiquetas de transición tienen una estructura que permite la interpretación de las transformaciones que se desean realizar sobre la infraestructura del ecosistema. Tienen que ser legibles y también fácilmente personalizables.

   Para las pruebas se han creado etiquetas fijas ligadas a los tipos de máquina que se han generado para disponer en el sistema.

   - `scale-up:1:api`

     Ha permitido crear una máquina virtual para el servicio api.

   - `scale-up:3:api`

     Ha permitido crear tres máquinas virtuales para el servicio api.

   - `scale-down:1:api`

     Ha permitido destruir una máquina virtual para el servicio api.

   - `scale-down:3:api`

     Ha permitido destruir tres máquinas virtuales para el servicio api.

   Se ha podido comprobar que la eficiencia de estas etiquetas de transición es dependiente del sistema que se pretende generar y el tipo de transiciones que necesita. Para un sistema básico como el que se plantea en el primer caso de uso, Massive Multiplayer Videogame (sección 4.1.1), puede servir con las etiquetas creadas en las pruebas,

sin embargo, un caso de uso más complejo, como el del tercer caso de uso, Job Processor (sección 4.1.3), puede necesitar que se habiliten más posibilidades a la hora de realizar transiciones, debido a que los trabajos pueden ser distribuidos en grupos con estructuras más complejas que requieran escalar múltiples servicios en entornos específicos.

Hace falta investigar más sobre este tema, valorando que las etiquetas sean personalizables y definibles, o la creación nuevos modelos de etiquetas que contemplen más casos. Tampoco se descarta la posibilidad de combinar un sistema fijo que contemple un elevado número de casos genéricos, pero que también exista dicha capacidad de crear etiquetas personalizadas para casos específicos.

3. **Carga de scripts de configuración sobre la infraestructura**

   Un elemento que no se ha tenido en cuenta inicialmente, pero que ha surgido a raíz de realizar ciertas pruebas, es la complejidad añadida a la configuración de la infraestructura instanciada dinámicamente.

   El sistema de generación de instancias de cada proveedor de infraestructura permite crear directamente contenedores Docker en sus propios gestores, y con esto reducir la necesidad de configuración, al no trabajar directamente con máquinas. Sin embargo, esto impide tener un control absoluto de los componentes del sistema y, en consecuencia, reduce la capacidad de generar un sistema autogestionado, ya que la gestión de los recursos pasaría a ser responsabilidad del proveedor de servicios. Pese a ello, es un sistema válido para ser utilizado con Cloud PIE y no se descarta como posibilidad a añadir en un futuro para el módulo de construcción.

   Debido a que el foco del proyecto se centra inicialmente en el aprovisionamiento de recursos de infraestructura para generar un entorno mejor adaptado, las pruebas han demostrado que es necesaria una configuración sobre las máquinas virtuales tras su instanciación. Los proveedores de servicios permiten la ejecución de scripts directa-

mente sobre estas máquinas tras su creación, y se han realizado las siguientes pruebas para verificar que el sistema puede funcionar con esta solución.

Se ha probado a cargar la máquina virtual creada con los siguientes scripts básicos para probar su funcionamiento:

- `touch test.txt`

  Se ha podido comprobar que el archivo ha sido creado en la máquina virtual.

- `echo "hola" >test.txt; mkdir testfolder; mv test.txt testfolder`

  Se ha podido comprobar que el archivo ha sido creado y metido en una carpeta con la información proporcionada.

Tras realizar las pruebas se ha verificado que esta funcionalidad es útil, pero se ha comprobado que hace falta la inclusión de un sistema que permita tener un mayor control sobre la configuración de la infraestructura, planteando la creación de un módulo específico que se encargue de ello con algunas herramientas que se han añadido al apartado de tecnologías, como son:

- Ansible (sección 5.5)
- Chef (sección 5.5)
- Kubernetes (sección 3.3.8)

La referencia al módulo de configuración se encuentra en la sección 5.5 de trabajo futuro.

4. **Capacidad del módulo de ser autónomo y ser utilizado externamente**

   Esta prueba se ha realizado con el objetivo de verificar que el módulo podría componerse en forma de microservicio y que el módulo principal, Brain, sea capaz de ejecutar tareas sobre él como un servicio externo.

Con este objetivo se han realizado peticiones con el módulo Brain a este módulo y se ha verificado que funciona correctamente.

Retainer

Para el módulo de persistencia se han realizado una serie de pruebas orientadas a comprobar si era capaz de guardar y, posteriormente, proporcionar la información guardada correctamente; así como verificar que esta información es suficiente y útil para el sistema.

Entre las pruebas que se han realizado están las siguientes:

1. **Correcto funcionamiento de la memoria a corto plazo**

   El objetivo de estas pruebas es verificar que la base de datos utilizada en la memoria a corto plazo, redis, está configurada correctamente y que guarda la información de la manera en que se desea. Además, se ha querido comprobar que el acceso posterior es rápido y que la información es accesible.

   Se han realizado las siguientes pruebas de guardado de datos:

```
1  retainer.save(
2    Memory::SHORT_TERM,
3    {
4      key: :machines,
5      value: [{
6        name: 'ubuntu-t2micro'
7        instance_type: 't2.micro',
8        image_id: 'ami-20ee5e5d'
9      }]
10   }
11 )
```

   Esto ha permitido guardar en la memoria a corto plazo del módulo retainer la información sobre la existencia de una máquina de tipo `t2.micro` y con sistema operativo

Ubuntu.

```
1   retainer.save(
2     Memory::SHORT_TERM,
3     {
4       key: :services,
5       value: [{
6         name: 'api'
7         machine_type: 'ubuntu-t2micro',
8         scale_count: '2'
9       }]
10    }
11  )
```

Con este método se ha podido guardar en la memoria a corto plazo la información sobre un servicio llamado `api`, que utiliza una máquina `ubuntu-t2micro` y tiene 2 instancias.

Posteriormente se ha realizado el acceso a los valores que se habían guardado previamente en la base de datos y se han obtenido de manera correcta.

2. **Correcto funcionamiento de la memoria a largo plazo**

   El objetivo de estas pruebas es verificar que la base de datos utilizada en la memoria a largo plazo, MongoDB, está configurada correctamente y que guarda la información de la manera en que se desea. Se ha querido comprobar que es capaz de trabajar con grandes cantidades de datos y realizar peticiones complejas sobre la base de datos.

   Se han realizado, entre otras, las siguientes pruebas de guardado de datos:

```
retainer.save(
  Memory::LONG_TERM,
  {
    collection: :data,
    value: {
      name: 'cpu_usage',
      value: 0.10056497175141259,
      created_at: '2018-07-08 10:00:23.120Z'
    }
  }
)
```

Con este comando se ha realizado el guardado de datos de uso de CPU.

```
retainer.save(
  Memory::LONG_TERM,
  {
    collection: :data,
    value: {
      name: 'disk_read_ops',
      value: 0.0,
      format: 'IOPS',
      created_at: '2018-07-08 10:00:23.120Z'
    }
  }
)
```

Con esta operación se ha realizado el guardado de datos de operaciones de lectura en disco, con el formato IOPS.

```
retainer.save(
  Memory::LONG_TERM,
  {
    collection: :data,
    value: {
      name: 'credits_consumed',
      value: 0.003866,
      created_at: '2018-07-08 10:00:23.120Z'
    }
  }
)
```

Finalmente, se ha probado a guardar otro valor que hace referencia al consumo de créditos.

Posteriormente se ha realizado el acceso a los valores que se habían guardado previamente en la base de datos y se han podido visualizar correctamente.

3. **Capacidad del módulo de ser autónomo y ser utilizado externamente**

   Esta prueba se ha realizado con el objetivo de verificar que el módulo podría componerse en forma de microservicio y que el módulo principal, Brain, sea capaz de ejecutar tareas sobre él como un servicio externo.

   Con este objetivo se han realizado peticiones con el módulo Brain a este módulo y se ha verificado que funciona correctamente.

Learner

Debido a la complejidad de implementar un modelo de inteligencia artificial que permita realizar aprendizaje máquina para la generación de nuevas reglas y evolucionar en la toma de decisiones, este sistema ha sido propuesto como trabajo futuro.

Sin embargo, debido a que ha sido considerado en el planteamiento inicial del proyecto y que la estructuración del sistema ha sido diseñada teniendo en cuenta la existencia de este

módulo, se ha implementado un módulo simple de aprendizaje que simula la devolución de respuestas basándose en los resultados históricos que se guardan en la base de datos.

Este módulo de aprendizaje no utiliza técnicas reales de machine learning, sino que devuelve automáticamente una serie de respuestas predefinidas que simulan los datos esperados en un sistema real.

La creación del módulo simulado ha permitido generar una estructura que, a efectos de las pruebas, permite comprobar si el módulo Learner está bien integrado en el sistema y si el concepto planteado en la arquitectura será útil cuando sea implementado con resultados reales.

Se han realizado las siguientes pruebas sobre este módulo:

1. **Obtención de nuevas reglas que permitan evolucionar el sistema**

   Debido a que se trata de un simulador, las reglas que han sido obtenidas son ficticias y no tienen relación con los datos sobre los sensores y las transiciones que se encuentran presentes en la base de datos de memoria a largo plazo del módulo Retainer. Sin embargo, estas pruebas han permitido comprobar que el módulo Learner es capaz de devolver una lista de reglas inferidas para su posterior uso.

2. **Capacidad del módulo de ser autónomo y ser utilizado externamente**

   Esta prueba se ha realizado con el objetivo de verificar que el módulo podría componerse en forma de microservicio y que el módulo principal, Brain, sea capaz de ejecutar tareas sobre él como un servicio externo.

   Con este objetivo se han realizado peticiones con el módulo Brain a este módulo y se ha verificado que funciona correctamente.

4.2.3 Pruebas en conjunto

Brain

El módulo maestro es el que controla todo el proyecto y, por eso, las pruebas realizadas sobre él han sido muy necesarias para verificar que el sistema funciona en conjunto, y que los ciclos de trabajo elegidos son correctos.

Aparte de realizar las pruebas unitarias por cada worker para verificar que estos funcionan correctamente, pueden ser utilizados externamente y son autónomos; se han realizado pruebas orientadas a verificar que cada una de las tareas o ciclos del módulo Brain actúan de la manera deseada y planteada en la arquitectura.

Por ello, se han realizado las siguientes pruebas:

1. **Inicialización del sistema por primera vez**

   Para probar esta funcionalidad se han creado unas plantillas simples para definir los tipos de máquina y también se han creado unas reglas básicas.

   Las plantillas que se han creado contienen la siguiente información:

   - Máquina virtual de tipo `t2.micro` que utiliza una imagen de sistema operativo Ubuntu.
   - Máquina virtual de tipo `t2.nano` con imagen de sistema operativo Ubuntu.
   - Servicio `api` que se lanza sobre máquinas de tipo `t2.micro`
   - Servicio `web` que se lanza sobre máquinas de tipo `t2.nano`

   Las reglas que se han utilizado indican las siguientes condiciones

   - Si el servicio `api` tiene un uso de CPU mayor que el 80 %, crear una nueva máquina `t2.micro` para este servicio.

- Si el servicio `web` tiene un uso de CPU mayor que el 80 %, crear una nueva máquina `t2.nano` para este servicio.
- Si el servicio `api` tiene un uso de CPU menor que el 20 % y tiene más de 1 máquina, eliminar una máquina `t2.micro`.
- Si el servicio `web` tiene un uso de CPU menor que el 20 % y tiene más de 1 máquina, eliminar una máquina `t2.micro`.
- Si estamos en un espacio temporal entre las 21:00 y las 23:00, aumentar las instancias de ambos servicios a un mínimo de 2 máquinas por servicio.

Con estos ficheros ya disponibles, se ha procedido a iniciar el ciclo, comenzando con la lectura de las plantillas de información del sistema.

Tras finalizar la lectura de las plantillas, el sistema ha procedido a comprobar si es necesario realizar la carga del último estado guardado o si, por el contrario, era la primera vez que se inicializaba. Al no existir ningún tipo de información en la base de datos, se ha procedido a continuar con una inicialización normal y se ha persistido la información guardada, creando el estado correspondiente en la base de datos.

Posteriormente, se ha procedido con la carga de las reglas, las cuales han pasado a formar parte del motor de reglas, y esto se ha verificado listando las reglas existentes.

Finalmente, se ha comprobado que el ciclo de inicio procede a ejecutar el ciclo de decisión tras verificar que toda la información es válida y que tiene los datos necesarios para continuar.

2. **Verificación del ciclo de decisión**

El ciclo de decisión es el principal proceso de ejecución del módulo Brain, y también del proyecto Cloud PIE, al tratarse de la funcionalidad que se ejecuta de manera reiterada para procesar toda la información y mutar el ecosistema.

Se han realizado pruebas en este módulo tras la inicialización del sistema de la prueba anterior, y se ha aprovechado para probar las funcionalidades de los workers en el ciclo principal.

En primer lugar, se ha realizado la petición de datos al módulo Gatherer, con la que se han obtenido una serie de valores que tienen el mismo formato y que muestran el estado actual del hardware de las instancias del sistema.

En este caso, al haberse inicializado el sistema por primera vez y no haberse instanciado aún ninguna máquina, no se han podido obtener resultados de la monitorización.

Esta información ha sido persistida en la memoria a largo plazo, en la base de datos MongoDB, gracias a la funcionalidad del módulo Retainer.

Posteriormente, se ha procesado esta información en el módulo Thinker, que utiliza el motor de reglas para tomar las decisiones sobre las mutaciones que se deben realizar en el sistema. El módulo Thinker ha proporcionado las etiquetas de transición correspondientes al cambio necesario en el sistema, según las reglas definidas.

Como esta prueba se ha realizado sobre las 21:30, y al no haber ninguna máquina presente, el sistema ha especificado la necesidad de instanciar un mínimo de dos máquinas por cada servicio mediante las siguientes etiquetas:

- `scale-to:2:api`
- `scale-to:2:web`

Se han guardado estos resultados en la memoria a largo plazo con el módulo Retainer.

Con las etiquetas de transición, se ha procedido a crear la nueva infraestructura necesaria a través del módulo Builder.

Como el sistema aún no había creado ninguna instancia, ha procedido a instanciar 4 máquinas, siendo 2 de cada tipo especificado previamente. Estas máquinas han

sido creadas correctamente y se ha verificado en el panel de control del proveedor de servicios Amazon AWS, como se puede observar en la figura 4.4.

Name	Instance ID	Instance Type	Availability Zone	Instance State
ubuntu	i-01533a9b0b8e2fce9	t2.nano	eu-west-3c	running
ubuntu	i-05f21e1f6b6f7d56f	t2.micro	eu-west-3c	running
ubuntu	i-06748b4fa501f328a	t2.micro	eu-west-3c	running
ubuntu	i-06d7a6c10b8e2d000	t2.nano	eu-west-3c	running

Figura 4.4: *Captura del panel de Amazon AWS*

Finalmente, se ha guardado un checkpoint del estado del sistema en la memoria a corto plazo.

En el checkpoint se indica la información correspondiente a la existencia de 2 máquinas de tipo `t2.micro` para el servicio `api` y 2 máquinas de tipo`t2.nano` para el servicio `web`.

Tras realizar todo el ciclo, el sistema ha programado la ejecución de un nuevo ciclo tras un tiempo de espera que había sido especificado en la plantilla de inicio.

Con estas pruebas se ha podido verificar que el sistema en conjunto funciona correctamente, que la ejecución del ciclo es correcta y secuencial; que el ciclo se ejecuta en bucle y que todos los módulos realizan las funciones esenciales para el funcionamiento de Cloud PIE.

3. **Verificación del ciclo de aprendizaje**

   Esta fase de pruebas ha sido utilizada para verificar que el módulo Brain ejecuta correctamente la fase del ciclo de aprendizaje y que realiza una correcta comunicación con los demás módulos presentes en este ciclo.

   En la primera fase de este ciclo, el módulo Brain ha utilizado el módulo Retainer para acceder a su base de datos MongoDB de la memoria a largo plazo, y esto le ha

permitido obtener una lista de valores, obtenidos por los sensores, que habían sido guardados previamente.

En la segunda fase del ciclo, se ha simulado el entrenamiento del modelo de aprendizaje en el módulo Learner. Sin embargo, no se ha realizado ninguna operación real de entrenamiento al tratarse de una simulación.

Posteriormente, se ha ejecutado la tercera fase de este ciclo, en la que se realiza un proceso de aprendizaje máquina sobre el módulo Learner. En ella, se ha simulado el tratamiento de los datos obtenidos previamente y se ha generado una lista de reglas, que ya habían sido definidas anteriormente en las pruebas unitarias de dicho módulo. Estas reglas han sido enviadas a la siguiente fase del ciclo.

En la fase final del ciclo, las reglas obtenidas a través del módulo Learner han sido importadas por el motor de reglas del módulo Thinker, lo que le ha permitido adquirir un conocimiento nuevo que podrá ser utilizado posteriormente para ejecutar nuevas decisiones sobre la evolución del ecosistema.

4. **Inicialización del sistema con datos guardados simulando un reinicio**

   Para esta pruebas, se han aprovechado los datos obtenidos previamente tras la inicialización del sistema por primera vez y la ejecución del ciclo de decisión y el ciclo de aprendizaje definidos en las pruebas anteriores.

   El sistema ha seguido el mismo patrón que en las pruebas de inicialización por primera vez pero, en el momento en el que ha tenido que verificar si era un reinicio, ha sido capaz de validar que en la base de datos existe información y ha procedido a cargar directamente el último estado en el que se encontraba; ignorando la información de inicio que había leído.

   Tras esto, ha procedido a cargar las reglas y, posteriormente, ha comenzado de nuevo el ciclo de decisión para adaptar el estado en el que se encontraba a la nueva situación obtenida a través de los sensores.

4.3 Resultados

Las pruebas no siempre han obtenido resultados favorables o han tenido el comportamiento esperado. Sin embargo, el sistema se ha comportado como había sido diseñado en la mayoría de los casos. Los resultados de las pruebas son los presentes en la tabla 4.1.

Prueba	Tipo	Resultado
Prueba Amazon AWS	Proveedor de infraestructura	Correcto
Prueba Microsoft Azure	Proveedor de infraestructura	Fallido
Prueba Google	Proveedor de infraestructura	Fallido
Prueba DigitalOcean	Proveedor de infraestructura	Fallido
Sensores hardware	Módulo Gatherer	Correcto
Abstracción con estructura MAD	Módulo Gatherer	Correcto
Sensores software	Módulo Gatherer	Simulado
Autonomía del módulo	Módulo Gatherer	Correcto
Ejecución de reglas	Módulo Thinker	Correcto
Importación de reglas dinámica	Módulo Thinker	Correcto
Autonomía del módulo	Módulo Thinker	Correcto
Mutación de la infraestructura	Módulo Builder	Correcto
Eficiencia etiquetas de transición	Módulo Builder	Fallido
Carga de scripts	Módulo Builder	Fallido
Autonomía del módulo	Módulo Builder	Correcto
Memoria a corto plazo funcional	Módulo Retainer	Correcto
Memoria a largo plazo funcional	Módulo Retainer	Correcto
Autonomía del módulo	Módulo Retainer	Correcto
Generación de reglas	Módulo Learner	Simulado
Autonomía del módulo	Módulo Learner	Correcto
Inicialización del sistema	Módulo Brain	Correcto
Ciclo de decisión	Módulo Brain	Correcto
Ciclo de aprendizaje	Módulo Brain	Simulado
Reinicio del sistema	Módulo Brain	Correcto

Cuadro 4.1: *Resultados de las pruebas*

Se han realizado un total de 24 pruebas que han permitido verificar el funcionamiento del sistema de forma unitaria y en conjunto, así como la correcta integración de varios proveedores de servicios.

Los resultados de las pruebas son, por tipo de prueba, los siguientes:

4.3.1 Pruebas con proveedores

Se han realizado pruebas con 4 tipos de proveedores distintos y el resultado ha sido: 1 correcta, 0 simuladas y 3 fallidas. Se presenta un 25 % de acierto.

4.3.2 Pruebas unitarias por módulo

Módulo Gatherer

Se han realizado 4 pruebas sobre el funcionamiento de este módulo y el resultado ha sido: 3 correctas, 1 simulada y 0 fallidas. Existe un 100 % de acierto, pero sólo un 75 % de las pruebas son reales.

Módulo Thinker

Se han realizado 3 pruebas sobre el funcionamiento de este módulo y el resultado ha sido: 3 correctas, 0 simuladas y 0 fallidas. Existe un 100 % de acierto.

Módulo Builder

Se han realizado 4 pruebas sobre el funcionamiento de este módulo y el resultado ha sido: 2 correctas, 0 simuladas y 2 fallidas. Existe un 50 % de acierto.

Módulo Retainer

Se han realizado 3 pruebas sobre el funcionamiento de este módulo y el resultado ha sido: 3 correctas, 0 simuladas y 0 fallidas. Existe un 100 % de acierto.

Módulo Learner

Se han realizado 2 pruebas sobre el funcionamiento de este módulo y el resultado ha sido: 1 correcta, 1 simulada y 0 fallidas. Existe un 100 % de acierto, pero sólo un 50 % de las pruebas son reales.

4.3.3 Pruebas en conjunto

Módulo Brain

Se han realizado 4 pruebas sobre el funcionamiento de este módulo y de su composición con el resto de módulos, y el resultado ha sido: 3 correctas, 1 simuladas y 0 fallidas. Existe un 100 % de acierto, pero sólo un 75 % de las pruebas son reales.

4.3.4 Resultados finales

En términos generales, de las 24 pruebas realizadas se extraen los siguientes datos:

- 16 pruebas han resultado correctas
- 3 pruebas han sido simuladas
- 5 pruebas han sido fallidas

Con estos valores se obtiene el resultado general de que un 79,17 % de las pruebas han resultado ser correctas, un 20,83 % han resultado fallidas y un 87,5 % de las pruebas totales se han basado en datos reales y no simulados, como se puede observar en la figura 4.5.

Figura 4.5: *Gráfico circular de los resultados de las pruebas*

4.4 Discusión

A partir de las pruebas realizadas y los resultados obtenidos se puede conseguir mucha información de interés para el proyecto, la cual se tendrá en cuenta en el planteamiento del trabajo futuro.

Los resultados que se han obtenido están basados en pruebas que no tienen el mismo peso, por lo que no se puede considerar que el porcentaje de pruebas correctas represente el valor de éxito. Sin embargo, a partir de las pruebas realizadas y de los resultados obtenidos, se puede extraer que hay unos módulos más desarrollados que otros y también se puede comprobar cuáles son las áreas en las que hace falta realizar más trabajo para mejorar el sistema.

Por un lado, queda claro que el sistema utilizado para la incorporación de las APIs de proveedores de servicios no es trivial; y se estima que sería favorable encontrar una librería que incluya este sistema implementado y que realice pruebas propias sobre esos proveedores, debido a la dificultad que existe para obtener créditos gratuitos que permitan probar el proyecto en la infraestructura de cada proveedor.

Por otro lado, hace falta ampliar el sistema de configuración de las máquinas virtuales, y sería interesante crear un nuevo módulo que se encargue específicamente de realizar esta configuración mediante el uso de librerías dedicadas.

Por último, las pruebas de concepto realizadas sobre la utilización de sensores software y el módulo Learner, de aprendizaje máquina, permiten observar que la integración de estas funcionalidades es muy útil para aumentar el valor de Cloud PIE y mejorar la eficiencia en sistemas complejos.

Capítulo 5

Trabajo futuro

En las primeras fases de este proyecto, con la investigación inicial y el estudio del estado del arte, los objetivos eran más difusos y era complicado realizar una definición precisa de la utilidad que se buscaba conseguir.

Poco a poco, se ha avanzado en el desarrollo del proyecto y se ha alcanzado una solución más definida y con unos objetivos fijos. Tras la implementación de la arquitectura y la realización de pruebas, se ha llegado a un estado en el que la herramienta funciona pero, sin embargo, no es todo lo eficiente que podría llegar a ser.

Se plantea la continuación del trabajo en el futuro, con el objetivo de mejorar el proyecto y de acercarse a una solución de mayor utilidad y, para ello, se presentan a continuación algunas ideas que podrían ser los siguientes focos de trabajo de Cloud PIE.

5.1 Múltiples proveedores de servicios

Uno de los objetivos de Cloud PIE es ser multiplataforma, permitiendo su uso en múltiples proveedores de servicios con el objetivo de obtener las ventajas particulares de cada uno de ellos y, a la vez, disponer de la libertad de elegir cuáles se adaptan mejor a las necesidades

particulares de cada servicio.

Entre los problemas detectados a la hora de realizar las pruebas se ha visto que, pese a que se han integrado librerías de dos proveedores y se ha intentado integrar otros dos más, sólo uno de ellos ha sido implementado de forma correcta y en total ha habido sólo un 25 % de eficacia a la hora de realizar estas integraciones.

El planteamiento inicial fue utilizar una librería que gestionara la creación de infraestructura en cloud y la obtención de datos de monitorización en múltiples proveedores y, por ello, se probó con Terraform. Esta librería finalmente fue descartada por diversos motivos, como se puede observar en la sección 2.2.1.

Sin embargo, gracias a la continuación de la investigación se ha encontrado que existen otras librerías con funcionalidad similar, como Abiquo, que pueden resultar de utilidad en el proyecto.

Por ello, se plantea la integración de otras librerías similares a Terraform[1], como Abiquo[2], y, en caso de que no sean de utilidad, focalizar parte del esfuerzo en realizar la integración correcta de múltiples APIs de proveedores de servicios.

5.2 Esquema de etiquetas eficiente

Las etiquetas de transición que habían sido planteadas en la arquitectura han mostrado ser de utilidad a la hora de crear infraestructura. Sin embargo, su utilidad es limitada y necesitan una reestructuración con el objetivo de mejorar la eficiencia de cara a sistemas más complejos.

Se plantea que este esquema sea realizado con objetos de tipo JSON o, en su defecto,

[1] https://www.terraform.io
[2] https://www.abiquo.com

generando algún tipo de estructura compleja que contemple la integración de argumentos personalizables, que permitan obtener un mayor control de las acciones que se pueden realizar a la hora de modificar la infraestructura.

5.3 Sensores software

La simulación de uso de sensores de software ha sido altamente satisfactoria y ha permitido visualizar su utilidad. Basar la gestión de la infraestructura únicamente en función de los datos obtenidos por el estado de las máquinas virtuales a nivel de hardware, es decir, datos de uso de la CPU, memoria RAM, etc., es muy limitante.

La utilización de sensores que detecten valores reales del estado de los servicios integrados en un sistema cloud permite un mayor acercamiento a las necesidades reales de dichos servicios, al ser capaz de disponer la infraestructura en función de los parámetros software con los que trabaja el sistema.

Por ello, se plantea como trabajo futuro la creación de una librería que pueda ser integrada en los servicios que son utilizados con Cloud PIE y una mayor investigación sobre este tema.

5.4 Machine Learning

Tras las pruebas realizadas con el módulo Learner, se ha llegado a la conclusión de que sería interesante trabajar más sobre este módulo, integrando una solución real de aprendizaje máquina que permita estudiar la gran cantidad de datos que son generados a través de la monitorización realizada por el módulo Gatherer.

Por un lado, se plantea la utilización de algoritmos de reglas de asociación [2] que per-

mitan detectar patrones en el comportamiento del sistema, como había sido planificado en la arquitectura de Cloud PIE en la sección 3.2.5.

Por otro lado, se propone el uso de otros sistemas de aprendizaje automático como, por ejemplo, la utilización de árboles de decisión [34] o de redes neuronales [19].

El objetivo de esta propuesta es realizar una mayor investigación sobre todas las posibilidades que ofrece la inteligencia artificial [35] para generar modelos predictivos y ver cuál sería su utilidad en el proyecto, estableciendo la posibilidad de combinar varios modelos para obtener mejores resultados.

5.5 Módulo Composer

La existencia de un módulo de configuración de máquinas ha sido propuesta previamente en este documento en la sección 4.3.2, debido a la gran necesidad que supone para gestionar el proceso de adecuación de las máquinas virtuales a los servicios que deben ser lanzados sobre ellas.

Este módulo debe encargarse de realizar un proceso de configuración sobre las instancias de los proveedores de infraestructura tras su lanzamiento, y debe ser capaz tanto de hacer que se comuniquen con otras instancias, como de gestionar el lanzamiento de servicios sobre estas máquinas.

Dada la dificultad que supone la configuración de sistemas avanzados mediante la utilización de scripts simples, se propone la utilización de librerías dedicadas que simplifican y facilitan en gran medida este proceso.

Algunos ejemplos de librerías propuestas son:

- Chef[3]
- Puppet[4]
- Ansible[5]

5.6 Actuación humana

En ciertas situaciones, las máquinas no son capaces de tomar las decisiones correctas de manera autónoma. Esto suele ocurrir cuando la fase de configuración de los sistemas, en este caso la configuración inicial, no ha sido realizada de la manera más eficiente, o cuando el sistema por sí mismo tiene faltas de diseño en la arquitectura.

No es raro que el sistema Cloud PIE requiera de ayuda humana para evolucionar correctamente, especialmente en las primeras etapas. Además, es necesario habilitar la posibilidad de que la configuración se pueda modificar fácilmente.

Por ello, se ha planteado como trabajo futuro la integración de un módulo que permita interactuar directamente con Cloud PIE, modificando las reglas, creando nueva infraestructura y añadiendo datos a la memoria a largo plazo para que sean tomados en consideración. Esto es importante, ya que los administradores cloud pueden necesitar que el sistema funcione de manera diferente de cara a un evento específico, así como descartar algunas de las soluciones que se han ido adquiriendo en la evolución del sistema.

[3] https://www.chef.io/chef
[4] https://puppet.com
[5] https://www.ansible.com

5.7 Panel web

Esta funcionalidad está muy relacionada con la necesidad de actuación humana, pero también permite otros usos. Se propone la existencia de un panel web debido a la gran ventaja que puede proporcionar disponer de una interfaz gráfica que permita ver el estado de Cloud PIE. Este panel web debería permitir visualizar las reglas que hay integradas en el sistema, el estado de los checkpoints, las máquinas y los servicios, los datos de monitorización que se han ido guardando, etc. Además, al tratarse de un panel gráfico, se pueden añadir estadísticas y gráficas de datos que proporcionen una visión clara del entorno.

La utilización de un panel web también puede ser de gran utilidad para un administrador de sistemas para que pueda adquirir la capacidad de control sobre éste mediante el uso de una interfaz simple.

5.8 Entorno de pruebas local

Se ha echado en falta la existencia de un entorno local que permita realizar las pruebas de Cloud PIE basándose en máquinas que no tienen gastos económicos relacionados con el consumo. El problema del uso de proveedores de infraestructura es que tienen un coste añadido a la hora de utilizarlos, y esto impide que las pruebas sean realizadas con sistemas complejos que utilicen un gran número de máquinas de manera continuada, ya que no existe una inversión sobre el proyecto que lo facilite.

Se propone generar un entorno local que utilice sistemas como OpenNebula[6] y OpenStack[7] para realizar las pruebas y mantener un sistema de integración continua [18] con las posibles modificaciones. Para ello, se plantea como posibilidad la utilización de ordenadores

[6] https://opennebula.org
[7] https://www.openstack.org

reducidos tipo Raspberry Pi[8], que permitan generar clústeres [1] para las pruebas con bajo coste y bajo consumo energético.

[8] https://www.raspberrypi.org

Capítulo 6

Conclusiones

La tecnología de cloud computing ha evolucionado mucho en los últimos años y esto ha hecho que los recursos de infraestructura sean escasos para la gran cantidad de servicios que hay en la nube. La solución propuesta por los propios proveedores de servicios se focaliza en mejorar la gestión de recursos hardware y no tiene tanto en cuenta las necesidades de los servicios en sí.

Los proveedores han creado SLAs que les permiten asegurar el aprovisionamiento de servicios, así como sistemas de reglas que permiten escalar automáticamente la infraestructura en caso de que sea necesario, pero esto no es suficiente.

Cloud PIE es un proyecto que busca solucionar este problema desde el punto de vista de los servicios, basándose en las necesidades particulares de cada uno de ellos para optimizar los recursos y adaptarse de manera más eficiente.

Se han creado muchas tecnologías cuyo objetivo es ayudar con este problema y, entre ellas, hay algunas que se parecen a la solución que propone Cloud PIE.

Sin embargo, Cloud PIE propone una nueva visión a la hora de gestionar la información, ya que no sólo utiliza datos obtenidos por la monitorización de los recursos hardware, sino

que también la obtiene de los servicios software. Esto es muy interesante, ya que mejora la capacidad de adaptación a los servicios teniendo en cuenta los parámetros reales con los que trabaja su software.

La información de estos recursos es gestionada por un motor de reglas que permite al proyecto procesar los datos y crear mutaciones sobre el ecosistema basadas en decisiones tomadas mediante técnicas eficientes de inteligencia artificial. Sin embargo, estas mutaciones pueden ser muy complejas y, para ello, se ha creado un sistema de etiquetas que las especifica, aunque es necesario mejorarlo.

Cloud PIE es capaz de realizar directamente modificaciones sobre la infraestructura de los proveedores cloud, pese a las dificultades que conlleva la integración de algunas de sus librerías y la gestión de todos sus sistemas que no utilizan los mismos esquemas de datos.

La propuesta de un sistema evolutivo de Cloud PIE muestra que las técnicas de aprendizaje máquina pueden ser aplicadas a campos como éste, permitiendo que las reglas que utiliza evolucionen y se adapten mejor al sistema.

Las pruebas realizadas sobre todos los módulos han demostrado que Cloud PIE puede ser una solución efectiva que reduzca la complejidad de tratar con sistemas de infraestructura cloud para asegurar la disponibilidad de los recursos; unificando todo en un ecosistema inteligente y multiplataforma.

Sin embargo, hace falta mucho más trabajo para que Cloud PIE se acerque a ser esa solución ideal para los administradores de sistemas, y existen muchas ideas para mejorarlo.

En definitiva, este trabajo es una primera aproximación a una solución que permite automatizar de forma inteligente y proactiva la gestión de recursos de infraestructura cloud, adaptándose a los servicios que hacen uso de Cloud PIE.

Chapter 6

Conclusions

Cloud computing technology has evolved greatly in recent years, and this has made infrastructure resources scarce for the large number of services in the cloud. The solution proposed by the service providers themselves is only focused on improving the management of hardware resources and does not take into account the actual needs of the services.

The providers have created SLAs that allow them to ensure the provisioning of services, as well as own systems of rules that allow the infrastructure to be scaled automatically if necessary, but this is not enough.

Cloud PIE is a project that seeks to solve this problem with the services in mind, based on the particular needs of each of them to optimize resources and adapt more efficiently.

Many technologies have been created whose goal is to solve this problem, and among them there are some that resemble Cloud PIE. However, Cloud PIE proposes a new vision of information management, since it does not only use the data obtained by monitoring hardware resources, but also obtains it from the services software. This is very interesting since it improves the ability to adapt to services taking into account the real parameters used by the software.

The information of these resources is used by a rules engine that allows the data to be processed and the mutation of the ecosystem, based on decisions made by efficient techniques of artificial intelligence. However, these mutations can be very complex and, for this matter, a transition specification labels system has been created, although it is necessary to improve it.

Cloud PIE is able to make direct changes to the infrastructure of cloud providers, despite the difficulties related to the integration of the API libraries and the differences between their data schemas.

The proposal of Cloud PIE's evolving system shows that machine learning techniques can be applied these fields of work, allowing the rules to evolve and adapt better to the system.

The tests carried out on all modules have shown that Cloud PIE can be an effective solution that reduces the complexity of dealing with cloud infrastructure systems and helps to ensure the availability of resources, unifying everything in a multiplatform intelligent ecosystem.

However, a lot more work is needed for Cloud PIE to come close to being that ideal solution for system administrators, and there are many ideas to improve it.

In short, this work is a first approach to a solution that allows to automate, in an intelligent and proactive way, the management of cloud infrastructure resources, adapting to the services that make use of Cloud PIE.

Bibliografía

[1] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkila, X. Wang, K. Hamily, et al. Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 170–175. IEEE, 2013.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[3] K. Banker. *MongoDB in action*. Manning Publications Co., 2011.

[4] L. Bass, I. Weber, and L. Zhu. *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[5] O. Ben-Kiki, C. Evans, and B. Ingerson. Yaml ain't markup language (yaml™) version 1.1. *yaml. org, Tech. Rep*, page 23, 2005.

[6] C. V. Blanco, E. Huedo, R. S. Montero, and I. M. Llorente. Dynamic provision of computing resources from grid infrastructures and cloud providers. In *Grid and Pervasive Computing Conference, 2009. GPC'09. Workshops at the*, pages 113–120. IEEE, 2009.

[7] M. K. Bowman-Amuah. Load balancer in environment services patterns, June 10 2003. US Patent 6,578,068.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66, 1997.

[9] Y. Brikman. *Terraform: Up and Running: Writing Infrastructure as Code.* .º'Reilly Media, Inc.", 2017.

[10] J. L. Carlson. *Redis in action.* Manning Publications Co., 2013.

[11] M. Chisholm. How to build a business rules engine. *San Francisco*, 2004.

[12] K. Chodorow. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage.* .º'Reilly Media, Inc.", 2013.

[13] R. Crespo Cepeda. Devops for dummies. 2017.

[14] D. Crockford. The application/json media type for javascript object notation (json). Technical report, 2006.

[15] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[16] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Readings in Artificial Intelligence and Databases*, pages 547–559. Elsevier, 1988.

[17] M. Fowler. *Domain-specific languages.* Pearson Education, 2010.

[18] M. Fowler and M. Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 122:14, 2006.

[19] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús. *Neural network design*, volume 20. Pws Pub. Boston, 1996.

[20] R. Han, J. Zhan, and J. V.-P. Luis. Sarp: Synopsis-based approximate request processing for low latency and small correctness loss in cloud online services. *International Journal of Parallel Programming*, 44(5):1054–1077, 2016.

[21] K. Hightower, B. Burns, and J. Beda. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. .°'Reilly Media, Inc.", 2017.

[22] E. Huedo, I. M. Llorente, et al. The gridway framework for adaptive scheduling and execution on grids. *Scalable Computing: Practice and Experience*, 6(3), 2005.

[23] A. Karasaridis, B. Rexroad, D. A. Hoeflin, et al. Wide-scale botnet detection and characterization. *HotBots*, 7:7–7, 2007.

[24] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4): 559–592, 2014.

[25] B. L. W. H. Y. Ma and B. Liu. Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining*, 1998.

[26] Y. Matsumoto. *Ruby in a Nutshell*. .°'Reilly Media, Inc.", 2002.

[27] P. Mell, T. Grance, et al. The nist definition of cloud computing. 2011.

[28] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[29] A. Moniruzzaman and S. A. Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[30] N. Naik. Building a virtual system of systems using docker swarm in multiple clouds. In *Systems Engineering (ISSE), 2016 IEEE International Symposium on*, pages 1–3. IEEE, 2016.

[31] N. M. Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[32] S. Newman. *Building microservices: designing fine-grained systems.* .°'Reilly Media, Inc.", 2015.

[33] M. Peacock. *Creating Development Environments with Vagrant.* Packt Publishing Ltd, 2015.

[34] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine learning*, pages 463–482. Springer, 1983.

[35] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[36] N. Sadashiv and S. D. Kumar. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science & Education (ICCSE), 2011 6th International Conference on*, pages 477–482. Citeseer, 2011.

[37] S. Tilkov and S. Vinoski. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010.

[38] G. Van Rossum and F. L. Drake. *Python language reference manual.* Network Theory United Kingdom, 2003.

[39] L. M. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[40] J. L. Vázquez-Poletti, R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente. Solidifying the foundations of the cloud for the next generation software engineering. *Journal of Systems and Software*, 86(9):2321–2326, 2013.

[41] J. L. Vazquez-Poletti, R. Moreno-Vozmediano, and I. M. Llorente. Admission control in the cloud: Algorithms for sla-based service model. In *Handbook of Research on Architectural Trends in Service-Driven Computing*, pages 701–717. IGI Global, 2014.

[42] J. L. Vázquez-Poletti, R. Moreno-Vozmediano, R. Han, W. Wang, and I. M. Llorente. Saas enabled admission control for mcmc simulation in cloud computing infrastructures. *Computer Physics Communications*, 211:88–97, 2017.

[43] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.